J Cryptol
https://doi.org/10.1007/s00145-018-9306-z

Journal of
CRYPTOLOGY

Locally Decodable and Updatable Non-malleable Codes and Their Applications

Dana Dachman-Soled*
University of Maryland, College Park, USA
danadach@ece.umd.edu

Feng-Hao Liu†
Florida Atlantic University, Boca Raton, USA
fenghao.liu@fau.edu

Elaine Shi‡
Cornell University, Ithaca, USA
runting@gmail.com

Hong-Sheng Zhou§
Virginia Commonwealth University, Richmond, USA
hszhou@vcu.edu

Communicated by Stefan Wolf.

Received 9 April 2017 / Revised 17 October 2018

Abstract. Non-malleable codes, introduced as a relaxation of error-correcting codes by Dziembowski, Pietrzak, and Wichs (ICS '10), provide the security guarantee that the message contained in a tampered codeword is either the same as the original message or is set to an unrelated value. Various applications of non-malleable codes have been discovered, and one of the most significant applications among these is the connection with tamper-resilient cryptography. There is a large body of work considering security against various classes of tampering functions, as well as non-malleable codes with enhanced features such as *leakage resilience*. In this work, we propose combining the concepts of *non-malleability*, *leakage resilience*, and *locality* in a coding scheme. The contribution of this work is threefold:

*Supported in part by NSF CAREER Award #CNS-1453045 and by a Ralph E. Powe Junior Faculty Enhancement Award.

†Supported in part by NSF award #CNS-1657040. This work was done, while the author was a postdoctoral researcher at the University of Maryland.

‡Supported in part by NSF award #CNS-1601879, a Packard Fellowship, and a DARPA Safeware Grant (subcontractor under IBM). This work was done, while the author was an assistant professor at the University of Maryland.

§Supported in part by NSF award #CNS-1801470.

Published online: 27 November 2018

1. As a conceptual contribution, we define a new notion of *locally decodable and updatable non-malleable code* that combines the above properties.
2. We present two simple and efficient constructions achieving our new notion with different levels of security.
3. We present an important application of our new tool—securing RAM computation against memory tampering and leakage attacks. This is analogous to the usage of traditional non-malleable codes to secure implementations in the *circuit* model against memory tampering and leakage attacks.

Keywords. Non-malleable codes, Leakage-resilient, Locally decodable.

1. Introduction

The notion of non-malleable codes was defined by Dziembowski et al. [29] as a relaxation of error-correcting codes. Informally, a coding scheme is **non-malleable** against a tampering function if by tampering with the codeword, the function can either keep the underlying message unchanged or change it to an unrelated message. Designing non-malleable codes is not only an interesting mathematical task, but also has important implications in cryptography; for example, Coretti et al. [19] showed an efficient construction of a multi-bit CCA secure encryption scheme from a single-bit one via non-malleable codes. Agrawal et al. [7] showed how to use non-malleable codes to build non-malleable commitments. Most notably, the notion has a deep connection with security against so-called *physical attacks*; indeed, using non-malleable codes to achieve security against physical attacks was the original motivation of the work [29]. Due to this important application, research on non-malleable codes has become an important agenda and drawn much attention in both coding theory and cryptography.

Briefly speaking, physical attacks target implementations of cryptographic algorithms beyond their input/output behavior. For example, researchers have demonstrated that leaking/tampering with sensitive secrets such as cryptographic keys, through timing channel, differential power analysis, and various other attacks, can be devastating [6,11,12,39,47,48,53], and therefore the community has focused on developing new mechanisms to defend against such strong adversaries [20–26,28,33–35,37,38,41–43,46,51,52,54]. Dziembowski et al. [29] showed a simple and elegant mechanism to secure implementations against memory tampering attacks by using non-malleable codes—instead of storing the secret (in the clear) on a device, one instead stores an encoding of the secret. The security of the non-malleable code guarantees that the adversary cannot learn more than what can be learnt via black box access to the device, even though the adversary may tamper with memory.

In a subsequent work, Liu and Lysyanskaya [50] extended the notion to capture **leakage resilience** as well—in addition to non-malleability, the adversary cannot learn anything about the underlying message even while obtaining partial leakage of the codeword. By using the approach outlined above, one can achieve security guarantees against both tampering and leakage attacks. In recent years, researchers have been studying various flavors of non-malleable codes; for example some work has focused on constructions against different classes of tampering functions, some has focused on different additional features (e.g., continual attacks, rates of the scheme), and some focused on other applications [3,7,15–17,27,30,32].

In this paper, we focus on another important feature inspired from the field of coding theory—**locality**. More concretely, we consider a coding scheme that is locally decodable and updatable. As introduced by Katz and Trevisan [44], local decodability means that in order to retrieve a portion of the underlying message, one does not need to read through the whole codeword. Instead, one can just read a few locations at the codeword. Similarly, local updatability means that in order to update some part of the underlying messages, one only needs to update some parts of the codeword. Locally decodable codes have many important applications in private information retrieval [18] and secure multi-party computation [40], and have deep connections with complexity theory; see [57]. Achieving local decodability and updatability simultaneously makes the task more challenging. Recently, Chandran et al. [13] constructed a locally decodable and updatable code in the setting of *error-correcting* codes. They also show an application to dynamic proofs of retrievability. Motivated by the above results, we further ask the following intriguing question:

> *Can we build a coding scheme enjoying all three properties, i.e., non-malleability, leakage resilience, and locality? If so, what are its implications in cryptography?*

Our Results In light of the above questions, our contribution is threefold:

- **(Notions)** We propose new notions that combine the concepts of non-malleability, leakage resilience, and locality in codes. First, we formalize a new notion of *locally decodable and updatable non-malleable codes* (against one-time attacks). Then, we extend this new notion to capture leakage resilience under continual attacks.
- **(Constructions)** We present two simple constructions achieving our new notions. The first construction is highly efficient—in order to decode (update) one block of the encoded messages, only *two* blocks of the codeword must be read (written)—but is only secure against *one-time* attacks. The second construction achieves security against *continual* attacks, while requiring $\log(n)$ number of reads (writes) to perform one decode (update) operation, where n is the number of blocks of the underlying message.
- **(Application)** We present an important application of our new notion—achieving tamper and leakage resilience in the random access machine (RAM) model. We first define a new model that captures tampering and leakage attacks in the RAM model, and then give a generic compiler that uses our new notion as a main ingredient. The compiled machine will be resilient to leakage and tampering on the random access memory. This is analogous to the usage of traditional non-malleable codes to secure implementations in the *circuit* model.

1.1. *Techniques*

In this section, we present a technical overview of our results.

Locally Decodable Non-malleable Codes Our first goal is to consider a combination of concepts of non-malleability and local decodability. Recall that a coding scheme is non-malleable with respect to a tampering function f if the decoding of the tampered codeword remains the same or becomes some unrelated message. To capture this idea,

the definition in the work [29] requires that there exists a simulator (with respect to such f) who outputs same^* if the decoding of the tampered codeword remains the same as the original one, or he outputs a decoded message, which is unrelated to the original one. In the setting of local decodability, we consider encodings of blocks of messages $M = (m_1, m_2, \ldots, m_n)$, and we are able to retrieve m_i by running $\text{DEC}^{\text{ENC}(M)}(i)$, where the decoding algorithm gets oracle access to the codeword.

The combination faces a subtlety that we cannot directly use the previous definition: suppose a tampering function f only modifies one block of the codeword, then it is likely that DEC remains unchanged for most places. (Recall a DEC will only read a few blocks of the codeword, so it may not detect the modification.) In this case, the (overall) decoding of $f(C)$ (i.e., $(\text{DEC}^{f(C)}(1), \ldots, \text{DEC}^{f(C)}(n))$) can be highly related to the original message, which intuitively means it is highly malleable.

To handle this issue, we consider a more fine-grained experiment. Informally, we require that for any tampering function f (within some class), there exists a simulator that computes a vector of decoded messages $\vec{m}^*$, a set of indices $\mathcal{I} \subseteq [n]$. Here $\mathcal{I}$ denotes the coordinates of the underlying messages that have been tampered with. If $\mathcal{I} = [n]$, then the simulator thinks that the decoded messages are $\vec{m}^*$, which should be unrelated to the original messages. On the other hand, if $\mathcal{I} \subsetneq [n]$, the simulator thinks that all the messages not in $\mathcal{I}$ remain unchanged, while those in $\mathcal{I}$ become $\perp$. This intuitively means the tampering function can do only one of the following cases:

1. It destroys a block (or blocks) of the underlying messages while keeping the other blocks unchanged, or
2. If it modifies a block of the underlying messages to some unrelated string, then it must have modified all blocks of the underlying messages to encodings of unrelated messages.

Our construction of locally decodable non-malleable code is simple—we use the idea similar to the *key encapsulation mechanism/data encapsulation mechanism (KEM/DEM) framework*. Let NMC be a regular non-malleable code, and $\mathcal{E}$ be a secure (symmetric key) authenticated encryption. Then to encode blocks of messages $M = (m_1, \ldots, m_n)$, we first sample a secret key sk of $\mathcal{E}$, and output $(\mathsf{NMC}.\text{ENC}(\mathsf{sk}), \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_1, 1), \ldots, \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_n, n))$. The intuition is clear: if the tampering function does not change the first block, then by security of the authenticated encryption, any modification of the rest will become $\perp$. (Note that here we include a tag of positions to prevent permutation attacks.) On the other hand, if the tampering function modified the first block, it must be decoded to an unrelated secret key sk'. Then by semantic security of the encryption scheme, the decoded values of the rest must be unrelated. The code can be updated locally: in order to update m_i to some m_i', one just need to retrieve the 1^{st} and $(i+1)^{\text{st}}$ blocks. Then he just computes a *fresh* encoding of $\mathsf{NMC}.\text{ENC}(\mathsf{sk})$ and the ciphertext $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_i')$, and writes back to the same positions.

Extensions to Leakage Resilience against Continual Attacks We further consider a notion that captures leakage attacks in the continual model. First we observe that suppose the underlying non-malleable code is also leakage resilient [50], the above construction also achieves one-time leakage resilience. Using the same argument of Liu and Lysyanskaya [50], if we can refresh the whole encoding, we can show that the

construction is secure against continual attacks. However, in our setting, refreshing the whole codeword is not counted as a solution since this is in the opposite of the spirit of our main theme—*locality*. The main challenge is how to refresh (update) the codeword locally while maintaining tamper and leakage resilience.

To capture the local refreshing and continual attacks, we consider a new model where there is an updater $\mathcal{U}$ who reads the whole underlying messages and decides how to update the codeword (using the local update algorithm). The updater is going to interact with the codeword in a continual manner, while the adversary can launch tampering and leakage attacks between two updates. To define security we require that the adversary cannot learn anything of the underlying messages via tampering and leakage attacks from the interaction.

We note that if there is no update procedure at all, then no coding scheme can be secure against continual leakage attacks if the adversary can learn the whole codeword bit by bit. In our model, the updater and the adversary take turns interacting with the codeword—the adversary tampers with and/or gets leakage of the codeword, and then the updater *locally* updates the codeword, and the process repeats. See Sect. 2 for the formal model.

Then we consider how to achieve this notion. First we observe that the construction above is not secure under continual attacks: suppose by leakage the adversary can get a full ciphertext $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_i, i)$ at some point, and then the updater updates the underlying message to m'_i. In the next round, the adversary can apply a *rewind attack* that modifies the codeword back with the old ciphertext. Under such attack, the underlying messages have been modified to some related messages. Thus the construction is not secure.

One way to handle this type of rewind attacks is to tie all the blocks of ciphertexts together with a "time stamp" that prevents the adversary from replacing the codeword with old ciphertexts obtained from leakage. A straightforward way is to hash all the blocks of encryptions using a collision-resistant hash function and also encode this value into the non-malleable code, i.e., $C = (\mathsf{NMC}.\textsc{enc}(\mathsf{sk}, v), \mathcal{E}.\mathsf{Encrypt}(1, m_1), \ldots, \mathcal{E}.\mathsf{Encrypt}(n, m_n))$, where $v = h(\mathcal{E}.\mathsf{Encrypt}(1, m_1), \ldots, \mathcal{E}.\mathsf{Encrypt}(n, m_n))$. Intuitively, suppose the adversary replaces a block $\mathcal{E}.\mathsf{Encrypt}(i, m_i)$ by some old ciphertexts, then it would be caught by the hash value v unless he tampered with the non-malleable code as well. But if he tampers with the non-malleable code, the decoding will be unrelated to sk, and thus the rest of ciphertexts become "un-decryptable." This approach prevents the rewind attacks, yet it does not preserve the local properties, i.e., to decode a block, one needs to check the consistency of the hash value v, which needs to read all the blocks of encryptions. To prevent the rewind attacks while maintaining local decodability/updatability, we use the Merkle tree technique, which allows local checks of consistency.

The final encoding outputs $(\mathsf{NMC}.\textsc{enc}(\mathsf{sk}, v), \mathcal{E}.\mathsf{Encrypt}(1, m_1), \ldots, \mathcal{E}.\mathsf{Encrypt}(n, m_n), T)$, where T is the Merkle tree of $(\mathcal{E}.\mathsf{Encrypt}(1, m_1), \ldots, \mathcal{E}.\mathsf{Encrypt}(n, m_n))$, and v is its root. (It can also be viewed as a hash value.) To decode a position i, the algorithm reads the 1^{st}, and the $(i+1)^{\text{st}}$ blocks together with a path in the tree. If the path is inconsistent with the root, then output $\perp$. To update, one only needs to re-encode the first block with a new root, and update the $(i+1)^{\text{st}}$ block and the tree. We note that Merkle tree allows local updates: if there is only one single change at a leaf, then one can compute the new root given only a path passing through the leaf and the

root. So the update of the codeword can be done locally by reading the 1^{st}, the $(i+1)^{\text{st}}$ blocks and the path. We provide a detailed description and analysis in Sect. 3.3.

Concrete Instantiations In our construction above, we rely on an underlying non-malleable code NMC against some class of tampering functions $\mathcal{F}$ and leakage resilient against some class of leakage functions $\mathcal{G}$. The resulting encoding scheme is a locally decodable and updatable coding scheme which is continual non-malleable against some class $\overline{\mathcal{F}}$ of tampering functions and leakage resilient against some class $\overline{\mathcal{G}}$ of leakage functions, where the class $\overline{\mathcal{F}}$ is determined by $\mathcal{F}$ and the class $\overline{\mathcal{G}}$ is determined by $\mathcal{G}$. In order to understand the relationship between these classes, it is helpful to recall the structure of the output of the final encoding scheme. The final encoding scheme will output $2n+1$ blocks $x_1, \ldots, x_{2n+1}$ such that the first block x_1 is encoded using the underlying non-malleable code NMC. As a first attempt, we can define $\overline{\mathcal{F}}$ to consist of tampering functions $f(x_1, \ldots, x_{2n+1}) = (f_1(x_1), f_2(x_2, \ldots, x_{2n+1}))$, where $f_1 \in \mathcal{F}$ and f_2 is any polynomial-sized circuit. However, it turns out that we are resilient against an even larger class of tampering functions! This is because the tampering function f_1 can actually depend on all the values $x_2, \ldots, x_{2n+1}$ of blocks $2, \ldots, (2n+1)$. Similarly, for the class of leakage functions, as a first attempt, we can define $\overline{\mathcal{G}}$ to consist of leakage functions $g(x_1, \ldots, x_{2n+1}) = (g_1(x_1), g_2(x_2, \ldots, x_{2n+1}))$, where $g_1 \in \mathcal{G}$ and g_2 is any polynomial-sized circuit. However, again we can achieve even more because the tampering function g_1 can actually depend on all the values $x_2, \ldots, x_{2n+1}$. For a formal definition of the classes of tampering and leakage functions that we handle, see Theorem 3.10.

Finally, we give a concrete example of what the resulting classes look like using the NMC construction of Liu and Lysyanskaya [50] as the building block. Recall that their construction achieves both tamper and leakage resilience for split-state functions. Thus, the overall tampering function f restricted in the first block (i.e., f_1) can be any (poly-sized) split-state function. On the other hand f restricted in the rest (i.e., f_2) can be any poly-sized function. The overall leakage function g restricted in the first block (i.e., g_1) can be a (poly-sized) length-bounded split-state function; g, on the other hand, can leak all the other parts. See Sect. 3.4 for more details.

Application to Tamper and Leakage-Resilient RAM Model of Computation Whereas regular non-malleable codes yield secure implementations against memory tampering in the circuit model, our new tool yields secure implementations against memory tampering (and leakage) in the RAM model.

In our RAM model, the data and program to be executed are stored in the random access memory. Through a CPU with a small number of (non-persistent) registers,[1] execution proceeds in clock cycles: in each clock-cycle memory addresses are read and stored in registers, a computation is performed, and the contents of the registers are written back to memory. In our attack model, we assume that the CPU circuitry (including the non-persistent registers) is secure—the computation itself is not subject to physical

[1] These non-persistent registers are viewed as part of the circuitry that stores some transient states, while the CPU is computing at each cycle. The number of these registers is small, and the CPU needs to erase the data in order to reuse them, so they cannot be used to store a secret key that is needed for a long term of computation.

attacks. On the other hand, the random access memory and the memory addresses are prone to leakage and tampering attacks. We remark that if the CPU has *secure* persistent registers that store a secret key, then the problem becomes straightforward: security can be achieved using encryption and authentication together with oblivious RAM [36]. We emphasize that in our model, persistent states of the CPU are stored in the memory, which are prone to leakage and tampering attacks. As our model allows the adversary to learn the access patterns the CPU made to the memory, together with the leakage and tampering power on the memory, the adversary can somewhat learn the messages transmitted over the *bus* or tamper with them (depending on the attack classes allowed on the memory). For simplicity of presentation, we do not define attacks on the bus, but just remark that these attacks can be implicitly captured by learning the access patterns and attacking the memory.[2]

In our formal modeling, we consider a next instruction function Π, a database D (stored in the random access memory), and an internal state (using the non-persistent registers). The CPU will interact (i.e., read/write) with the memory based on Π, while the adversary can launch tamper and leakage attacks during the interaction.

Our compiler is very simple, given the ORAM technique and our new codes as building blocks. Informally speaking, given any next instruction function Π and database D, we first use ORAM technique to transform them into a next instruction function $\widetilde{\Pi}$ and a database $\widetilde{D}$. Next, we use our local non-malleable code (ENC, DEC, UPDATE) to encode $\widetilde{D}$ into $\widehat{D}$; the compiled next instruction function $\widehat{\Pi}$ does the following: run $\widetilde{\Pi}$ to compute the next "virtual" read/write instruction, and then run the local decoding or update algorithms to perform the physical memory access.

Intuitively, the inner ORAM protects leakage of the address patterns, and the outer local non-malleable codes prevent an attacker from modifying the contents of memory to some *different* but *related* value. Since at each cycle the CPU can only read and write at a small number of locations of the memory, using regular non-malleable codes does not work. Our new notion of locally decodable and updatable non-malleable codes exactly solves these issues!

1.2. *Related Work*

Different flavors of non-malleable codes were studied [2,3,7,8,15–17,27,29,30,32,50]. We can use these constructions to secure implementations against memory attacks in the circuit model and also as our building block for the locally decodable/updatable non-malleable codes. See also Sect. 3.4 for further exposition.

Securing circuits or CPUs against physical attacks is an important task, but out of the scope of this paper. Some partial results can be found in previous work [20,21,23–26,28,33,34,37,38,41–43,46,49,51,52,54–56].

In an independent and concurrent work, Faust et al. [31] also considered securing RAM computation against tampering and leakage attacks. We note that both their model and techniques differ considerably from ours. In the following, we highlight some of these differences. The main focus of [31] is constructing RAM compilers for keyed functions,

[2] There are some technical subtleties to simulate all leakage/tampering attacks on the values passing the bus using memory attacks (and addresses). We defer the rigorous treatment to future work.

denoted $\mathcal{G}_{\mathsf{K}}$, to allow secure RAM emulation of these functions in the presence of leakage and tampering. In contrast, our work focuses on constructing compilers that transform any dynamic RAM machine into a RAM machine secure against leakage and tampering. Due to this different perspective, our compiler explicitly utilizes an underlying ORAM compiler, while they assume that the memory access pattern of input function $\mathcal{G}$ is independent of the secret state K (e.g., think of $\mathcal{G}$ as the circuit representation of the function). In addition to the split-state tampering and leakage attacks considered by both papers, [31] do not assume that memory can be overwritten or erased, but require the storage of a tamper-proof program counter. With regard to techniques, they use a stronger version of non-malleable codes in the split-state setting (called continual non-malleable codes [30]) for their construction. Finally, in their construction, each memory location is encoded using an expensive non-malleable encoding scheme, while in our construction, non-malleable codes are used only for a small portion of the memory, while highly efficient symmetric key authenticated encryption is used for the remainder.

1.3. *Subsequent Work*

Subsequent to the publication of our work, a significant progress has been made on constructions of improved (split-state) non-malleable codes (cf. [1,4,5]) and these constructions can then be plugged into our construction which generically constructs locally decodable and updatable non-malleable codes from an underlying (regular) non-malleable codes.

Finally, Chandran et al. [14] presented constructions of *information-theoretic* locally decodable and updatable non-malleable codes, which do not require computational assumptions. Their constructions, however, do not extend to the continual leakage setting and so should be compared with our one-time construction (see Sect. 3.2). In this setting, their constructions require non-constant locality, whereas our constructions achieve constant locality.

2. Locally Decodable and Updatable Non-malleable Codes

In this section, we first review the concepts of non-malleable (leakage resilient) codes. Then we present our new notion that combines non-malleability, leakage resilience, and locality.

2.1. *Preliminary*

Definition 2.1. (*Coding Scheme*) Let Σ, $\hat{\Sigma}$ be sets of strings, and $\kappa, \hat{\kappa} \in \mathbb{N}$ be some parameters. A coding scheme consists of two algorithms (ENC, DEC) with the following syntax:

- The encoding algorithm (*perhaps randomized*) takes input a block of message in Σ and outputs a codeword in $\hat{\Sigma}$.
- The decoding algorithm takes input a codeword in $\hat{\Sigma}$ and outputs a block of message in Σ.

We require that for any message $m \in \Sigma$, $\Pr[\text{DEC}(\text{ENC}(m)) = m] = 1$, where the probability is taken over the choice of the encoding algorithm. In binary settings, we often set $\Sigma = \{0,1\}^\kappa$ and $\hat{\Sigma} = \{0,1\}^{\hat{\kappa}}$.

Definition 2.2. (*Non-malleability* [29]) Let k be the security parameter, and $\mathcal{F}$ be some family of functions. For each function $f \in \mathcal{F}$, and $m \in \Sigma$, define the tampering experiment:

$$\mathbf{Tamper}_m^f \stackrel{\text{def}}{=} \left\{ \begin{array}{c} c \leftarrow \text{ENC}(m), \tilde{c} := f(c), \tilde{m} := \text{DEC}(\tilde{c}). \\ Output: \ \tilde{m}. \end{array} \right\},$$

where the randomness of the experiment comes from the encoding algorithm. We say a coding scheme (ENC, DEC) is non-malleable with respect to $\mathcal{F}$ if for each $f \in \mathcal{F}$, there exists a PPT simulator $\mathcal{S}$ such that for any message $m \in \Sigma$, we have

$$\begin{aligned} &\mathbf{Tamper}_m^f \approx \mathbf{Ideal}_{\mathcal{S},m} \\ &\stackrel{\text{def}}{=} \left\{ \begin{array}{c} \tilde{m} \cup \{\mathsf{same}^*\} \leftarrow \mathcal{S}^{f(\cdot)}. \\ Output: m \text{ if that is } \mathsf{same}^*; \text{ otherwise } \tilde{m}. \end{array} \right\} \end{aligned}$$

Here the indistinguishability can be either computational or statistical.

We can extend the notion of non-malleability to leakage resilience (simultaneously) as the work of Liu and Lysyanskaya [50].

Definition 2.3. (*Non-malleability and Leakage Resilience* [50]) Let k be the security parameter, $\mathcal{F}$, $\mathcal{G}$ be some families of functions. For each function $f \in \mathcal{F}$, $g \in \mathcal{G}$, and $m \in \Sigma$, define the tamper-leak experiment:

$$\mathbf{TamperLeak}_m^{f,g} \stackrel{\text{def}}{=} \left\{ \begin{array}{c} c \leftarrow \text{ENC}(m), \tilde{c} := f(c), \tilde{m} := \text{DEC}(\tilde{c}). \\ Output: \ (\tilde{m}, g(c)). \end{array} \right\},$$

where the randomness of the experiment comes from the encoding algorithm. We say a coding scheme (ENC, DEC) is non-malleable and leakage resilience with respect to $\mathcal{F}$ and $\mathcal{G}$ if for any $f \in \mathcal{F}$, $g \in \mathcal{G}$, there exists a PPT simulator $\mathcal{S}$ such that for any message $m \in \Sigma$, we have

$$\begin{aligned} &\mathbf{TamperLeak}_m^{f,g} \approx \mathbf{Ideal}_{\mathcal{S},m} \\ &\stackrel{\text{def}}{=} \left\{ \begin{array}{c} (\tilde{m} \cup \{\mathsf{same}^*\}, \ell) \leftarrow \mathcal{S}^{f(\cdot),g(\cdot)}. \\ Output: (m, \ell) \text{ if that is } \mathsf{same}^*; \text{ otherwise } (\tilde{m}, \ell). \end{array} \right\} \end{aligned}$$

Here the indistinguishability can be either computational or statistical.

2.2. *New Definitions: Codes with Local Properties*

In this section, we consider coding schemes with extra *local* properties—decodability and updatability. Intuitively, this gives a way to encode blocks of messages, such that

in order to decode (retrieve) a single block of the messages, one only needs to read a small number of blocks of the codeword; similarly, in order to update a single block of the messages, one only needs to update a few blocks of the codeword.

Definition 2.4. (*Locally Decodable and Updatable Code*) Let Σ, $\hat{\Sigma}$ be sets of strings, and $n, \hat{n}, p, q$ be some parameters. An $(n, \hat{n}, p, q)$ locally decodable and updatable coding scheme consists of three algorithms (ENC, DEC, UPDATE) with the following syntax:

- The encoding algorithm ENC (*perhaps randomized*) takes input an n-block (in Σ) message and outputs an $\hat{n}$-block (in $\hat{\Sigma}$) codeword.
- The (local) decoding algorithm DEC takes input an index in $[n]$, reads at most p blocks of the codeword, and outputs a block of message in Σ. The overall decoding algorithm simply outputs $(\text{DEC}(1), \text{DEC}(2), \ldots, \text{DEC}(n))$.
- The (local) updating algorithm UPDATE (*perhaps randomized*) takes inputs an index in $[n]$ and a string in $\Sigma \cup \{\epsilon\}$ and reads/writes at most q blocks of the codeword. Here the string ϵ denotes the procedure of refreshing without changing anything.

Let $C \in \hat{\Sigma}^{\hat{n}}$ be a codeword. For convenience, we denote DEC^C, UPDATE^C as the processes of reading/writing individual block of the codeword, i.e., the codeword oracle returns or modifies individual block upon a query. Here we view C as a random access memory where the algorithms can read/write to the memory C at individual different locations.

Remark 2.5. Throughout this paper, we only consider non-adaptive decoding and updating, which means the algorithms DEC and UPDATE compute all their queries at the same time before seeing the answers, and the computation only depends on the input i (the location). In contrast, an adaptive algorithm can compute a query based on the answer from previous queries. After learning the answer to such query, then it can make another query. We leave it as an interesting open question to construct more efficient schemes using adaptive queries.

Then we define the requirements of the coding scheme.

Definition 2.6. (*Correctness*) An $(n, \hat{n}, p, q)$ locally decodable and updatable coding scheme (with respect to Σ, $\hat{\Sigma}$) satisfies the following properties. For any message $M = (m_1, m_2, \ldots, m_n) \in \Sigma^n$, let $C = (c_1, c_2, \ldots, c_{\hat{n}}) \leftarrow \text{ENC}(M)$ be a codeword output by the encoding algorithm. Then we have:

- for any index $i \in [n]$, $\Pr[\text{DEC}^C(i) = m_i] = 1$, where the probability is over the randomness of the encoding algorithm.
- for any update procedure with input $(j, m') \in [n] \times \Sigma \cup \{\epsilon\}$, let C' be the resulting codeword by running $\text{UPDATE}^C(j, m')$. Then we have $\Pr[\text{DEC}^{C'}(j) = m'] = 1$, where the probability is over the encoding and update procedures. Moreover, the decodings of the other positions remain unchanged.

Remark 2.7. The correctness definition can be directly extended to handle any sequence of updates.

Next, we define several flavors of security about non-malleability and leakage resilience.

One-time Non-malleability First we consider one-time non-malleability of locally decodable codes, i.e., the adversary only tampers with the codeword once. This extends the idea of the non-malleable codes (as in Definition 2.2). As discussed in introduction, we present the following definition to capture the idea that the tampering function can only do either of the following cases:

- It destroys a block (or blocks) of the underlying messages while keeping the other blocks unchanged, or
- If it modifies a block of the underlying messages to some unrelated string, then it must have modified all blocks of the underlying messages to encodings of unrelated messages.

Definition 2.8. (*Non-malleability of Locally Decodable Codes*) An $(n, \hat{n}, p, q)$-locally decodable coding scheme with respect to $\Sigma, \hat{\Sigma}$ is non-malleable against the tampering function class $\mathcal{F}$ if for all $f \in \mathcal{F}$, there exists some simulator $\mathcal{S}$ such that for any $M = (m_1, \ldots, m_n) \in \Sigma^n$, the experiment $\mathbf{Tamper}_M^f$ is (computationally) indistinguishable to the following ideal experiment $\mathbf{Ideal}_{\mathcal{S},M}$:

- $(\mathcal{I}, \vec{m}^*) \leftarrow \mathcal{S}(1^k)$, where $\mathcal{I} \subseteq [n]$, $\vec{m}' \in \Sigma^n$. (Intuitively $\mathcal{I}$ means the coordinates of the underlying message that have been tampered with.)
- If $\mathcal{I} = [n]$, define $\vec{m} = \vec{m}^*$; otherwise set $\vec{m}|_{\mathcal{I}} = \perp, \vec{m}|_{\bar{\mathcal{I}}} = M|_{\bar{\mathcal{I}}}$, where $\vec{x}|_{\mathcal{I}}$ denotes the coordinates $\vec{x}[v]$ where $v \in \mathcal{I}$, and the bar denotes the complement of a set.
- The experiment outputs $\vec{m}$.

Remark 2.9. Here we make two remarks about the definition:

1. In the one-time security definition, we do not consider the update procedure. In the next paragraph when we define continual attacks, we will handle the update procedure explicitly.
2. One-time leakage resilience of locally decodable codes can be defined in the same way as Definition 2.3.

Security Against Continual Attacks In the following, we extend the security to handle continual attacks. Here we consider a third party called *updater*, who can read the underlying messages and decide how to update the codeword. Our model allows the adversary to learn the location that the updater updated the messages, so we also allow the simulator to learn this information. This is without loss of generality if the leakage class $\mathcal{G}$ allows it, i.e., the adversary can query some $g \in \mathcal{G}$ to figure out what location was modified. On the other hand, the updater does not tell the adversary what content was encoded of the updated messages, so the simulator needs to simulate the view without such information. We can think of the updater as an honest user interacting with the codeword (read/write). The security intuitively means that even if the adversary can launch tampering and leakage attacks when the updater is interacting with the codeword, the adversary cannot learn anything about the underlying encoded messages (or the updated messages during the interaction).

Our continual experiment consists of rounds: in each round the adversary can tamper with the codeword and get partial information. At the end of each round, the updater will run UPDATE, and the codeword will be somewhat updated and refreshed. We note that if there is no refreshing procedure, then no coding scheme can be secure against continual leakage attack even for one-bit leakage at a time,[3] so this property is necessary. Our concept of "continuity" is different from that of Faust et al. [30], who considered continual attacks on the same original codeword. (The tampering functions can be chosen adaptively.) Our model does not allow this type of "resetting attacks." Once a codeword has been modified to $f(C)$, the next tampering function will be applied on $f(C)$.

We remark that the one-time security can be easily extended to the continual case (using a standard hybrid argument) if the update procedure re-encodes the *whole* underlying messages (c.f. see the results in the work [50]). However, in the setting above, we emphasize on the *local property*, so this approach does not work. How to do a local update while maintaining tamper and leakage resilience makes the continual case challenging!

Definition 2.10. (*Continual Tampering and Leakage Experiment*) Let k be the security parameter, $\mathcal{F}, \mathcal{G}$ be some families of functions. Let (ENC, DEC, UPDATE) be an $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$. Let $\mathcal{U}$ be an updater that takes input a message $M \in \Sigma^n$ and outputs an index $i \in [n]$ and $m \in \Sigma$. Then for any blocks of messages $M = (m_1, m_2, \ldots, m_n) \in \Sigma^n$, and any (non-uniform) adversary $\mathcal{A}$, any updater $\mathcal{U}$, define the following continual experiment $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M}$:

- The challenger first computes an initial encoding $C^{(1)} \leftarrow \text{ENC}(M)$.
- Then the following procedure repeats, at each round j, let $C^{(j)}$ be the current codeword and $M^{(j)}$ be the underlying message:
  - $\mathcal{A}$ sends either a tampering function $f \in \mathcal{F}$ and/or a leakage function $g \in \mathcal{G}$ to the challenger.
  - The challenger replaces the codeword with $f(C^{(j)})$ or sends back a leakage $\ell^{(j)} = g(C^{(j)})$.
  - We define $\vec{m}^{(j)} \stackrel{\text{def}}{=} \left(\text{DEC}^{f(C^{(j)})}(1), \ldots, \text{DEC}^{f(C^{(j)})}(n)\right)$.
  - Then the updater computes $(i^{(j)}, m) \leftarrow \mathcal{U}(\vec{m}^{(j)})$ for the challenger.
  - Then the challenger runs $\text{UPDATE}^{f(C^{(j)})}(i^{(j)}, m)$ and sends the index $i^{(j)}$ to $\mathcal{A}$.
  - $\mathcal{A}$ may terminate the procedure at any point.
- Let t be the total number of rounds above. At the end, the experiment outputs

$$\left(\ell^{(1)}, \ell^{(2)}, \ldots, \ell^{(t)}, \vec{m}^{(1)}, \ldots, \vec{m}^{(t)}, i^{(1)}, \ldots, i^{(t)}\right).$$

Definition 2.11. (*Non-malleability and Leakage Resilience against Continual Attacks*) An $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$ is continual non-malleable against $\mathcal{F}$ and leakage resilient against $\mathcal{G}$ if for all PPT (non-

[3] If there is no refreshing procedure, then the adversary can eventually learn the whole codeword bit by bit by leakage. Thus he can learn the underlying message.

uniform) adversaries $\mathcal{A}$, and PPT updaters $\mathcal{U}$, there exists some PPT (non-uniform) simulator $\mathcal{S}$ such that for any $M = (m_1, \ldots, m_n) \in \Sigma^n$, $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M}$ is (computationally) indistinguishable to the following ideal experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$:

- The experiment proceeds in rounds. Let $M^{(1)} = M$ be the initial message.
- At each round j, the experiment runs the following procedure:
  - At the beginning of each round, $\mathcal{S}$ outputs $(\ell^{(j)}, \mathcal{I}^{(j)}, \vec{w}^{(j)})$, where $\mathcal{I}^{(j)} \subseteq [n]$.
  - Define

$$\vec{m}^{(j)} = \begin{cases} \vec{w}^{(j)} & \text{if } \mathcal{I}^{(j)} = [n] \\ \vec{m}^{(j)}|_{\mathcal{I}^{(j)}} := \perp, \vec{m}^{(j)}|_{\bar{\mathcal{I}}^{(j)}} := M^{(j)}|_{\bar{\mathcal{I}}^{(j)}} & \text{otherwise,} \end{cases}$$

    where $\vec{x}|_{\mathcal{I}}$ denotes the coordinates $\vec{x}[v]$ where $v \in \mathcal{I}$, and the bar denotes the complement of a set.
  - The updater runs $(i^{(j)}, m) \leftarrow \mathcal{U}(\vec{m}^{(j)})$ and sends the index $i^{(j)}$ to the simulator. Then the experiment updates $M^{(j+1)}$ as follows: set $M^{(j+1)} := M^{(j)}$ for all coordinates except $i^{(j)}$, and set $M^{(j+1)}[i^{(j)}] := m$.
- Let t be the total number of rounds above. At the end, the experiment outputs

$$\left(\ell^{(1)}, \ell^{(2)}, \ldots, \ell^{(t)}, \vec{m}^{(1)}, \ldots, \vec{m}^{(t)}, i^{(1)}, \ldots, i^{(t)}\right).$$

2.3. *Strong Non-malleability*

Here we first recall the strong non-malleability notion originally defined by Dziembowski et al. [29]. Then we define strong non-malleability against one-time and continual attacks, respectively. We remark that our constructions in Sect. 3 can achieve the stronger notion of non-malleability if the underlying non-malleable code is the stronger one (see Remark 3.15).

Definition 2.12. (*Strong Non-malleability* [29]) Let k be the security parameter, $\mathcal{F}$ be some family of functions. For each function $f \in \mathcal{F}$, and $m \in \Sigma$, define the tampering experiment

$$\mathbf{StrongNM}_m^f \stackrel{\text{def}}{=} \left\{ \begin{array}{c} c \leftarrow \text{ENC}(m), \tilde{c} := f(c), \tilde{m} := \text{DEC}(\tilde{c}) \\ Output: \ \mathsf{same}^* \text{ if } \tilde{c} = c, \text{ and } \tilde{m} \text{ otherwise.} \end{array} \right\}$$

The randomness of this experiment comes from the randomness of the encoding algorithm. We say that a coding scheme (ENC, DEC) is strong non-malleable with respect to the function family $\mathcal{F}$ if for any $m, m' \in \Sigma$ and for each $f \in \mathcal{F}$, we have:

$$\{\mathbf{StrongNM}_m^f\}_{k\in\mathbb{N}} \approx \{\mathbf{StrongNM}_{m'}^f\}_{k\in\mathbb{N}}$$

where $\approx$ can refer to statistical or computational indistinguishability.

One-time security Strong Non-malleability against one-time physical attacks is defined as follows.

Definition 2.13. (*Strong Non-malleability of Locally Decodable Codes*) Let k be the security parameter, $\mathcal{F}$ be some family of functions. For each function $f \in \mathcal{F}$, and $M = (m_1, m_2, \ldots, m_n) \in \Sigma^n$, define the tampering experiment

$$\mathbf{StrongNM}_M^f \stackrel{\text{def}}{=} \left\{ \begin{array}{l} C \leftarrow \text{ENC}(M), \tilde{C} = f(C), \tilde{m}_i = \text{DEC}^{\tilde{C}}(i) \text{ for } i \in [n]. \\ \text{If } \exists\, i \text{ such that } \tilde{m}_i \neq \perp \,\&\, \tilde{C} \text{ and } C \text{ are not identical for all queries by } \text{DEC}(i), \\ \quad \text{then output: } (\tilde{m}_1, \tilde{m}_2, \ldots, \tilde{m}_n). \\ \text{Else, set } m'_i = \mathsf{same}^* \text{ if } C \text{ and } \tilde{C} \text{ are identical for all queries of } \text{DEC}(i); \\ \quad \text{otherwise } m'_i = \perp. \text{ Then output: } (m'_1, m'_2, \ldots, m'_n). \end{array} \right\}$$

The randomness of this experiment comes from the randomness of the encoding and decoding algorithms.

We say that a locally decodable coding scheme (ENC, DEC, UPDATE) is strong non-malleable against the function class $\mathcal{F}$ if for any $M, M' \in \Sigma^n$ and for any $f \in \mathcal{F}$, we have:

$$\{\mathbf{StrongNM}_M^f\}_{k\in\mathbb{N}} \approx \{\mathbf{StrongNM}_{M'}^f\}_{k\in\mathbb{N}}$$

where $\approx$ can refer to statistical or computational indistinguishability.

Continual security Strong Non-malleability against continual physical attacks is defined as follows.

Definition 2.14. (*Strong Continual Tampering and Leakage Experiment*) Let k be the security parameter, $\mathcal{F}, \mathcal{G}$ be some families of functions. Let (ENC, DEC, UPDATE) be an $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$. Let $\mathcal{U}$ be an updater that takes input a message M and outputs an index $i \in [n]$ and $m \in \Sigma$. Then for any blocks of messages $M = (m_1, m_2, \ldots, m_n) \in \Sigma^n$, and any (non-uniform) adversary $\mathcal{A}$, any updater $\mathcal{U}$, define the following experiment $\mathbf{StrongTL}_{\mathcal{A},\mathcal{U},M}$:

- The challenger first computes an initial encoding $C^{(1)} \leftarrow \text{ENC}(M)$.
- Then the following procedure repeats, at each round j, let $C^{(j)}$ be the current codeword and $M^{(j)}$ be the underlying message:
  - $\mathcal{A}$ sends either a tampering function $f \in \mathcal{F}$ and/or a leakage function $g \in \mathcal{G}$ to the challenger.
  - The challenger replaces the codeword with $f(C^{(j)})$ or sends back a leakage $\ell^{(j)} = g(C^{(j)})$.
  - Then we define $\vec{m}^{(j)}$ for the following two conditions:
    - If there exists i such that $\text{DEC}^{f(C^{(j)})}(i) \neq \perp$ and $C^{(j)}$ and $f(C^{(j)})$ are not identical for all queries from $\text{DEC}(i)$, then set $\vec{m}^{(j)} = (\text{DEC}^{f(C^{(j)})}(1), \ldots, \text{DEC}^{f(C^{(j)})}(n))$.
    - Else, for $i \in [n]$, let $m'_i = \mathsf{same}^*$ if $f(C^{(j)})$ and $C^{(j)}$ are identical for all queries of $\text{DEC}(i)$, otherwise $m'_i = \perp$. Then set $\vec{m}^{(j)} = (m'_1, m'_2, \ldots, m'_n)$.

  - Then the updater sends $(i^{(j)}, m) \leftarrow \mathcal{U}(M^{(j)})$ to the challenger, and the challenger runs $\text{UPDATE}^{f(C^{(j)})}(i^{(j)}, m)$ and sends the index $i^{(j)}$ to $\mathcal{A}$.
  - $\mathcal{A}$ may terminate the procedure at any point.
- Let t be the total number of rounds above. At the end, the experiment outputs

$$\left(\ell^{(1)}, \ell^{(2)}, \ldots, \ell^{(t)}, \vec{m}^{(1)}, \ldots, \vec{m}^{(t)}, i^{(1)}, \ldots, i^{(t)}\right).$$

Definition 2.15. (*Strong Non-malleability and Leakage Resilience against Continual Attacks*) An $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$ is strong continual non-malleable against $\mathcal{F}$ and leakage resilient against $\mathcal{G}$ if for all PPT (non-uniform) adversaries $\mathcal{A}$, any PPT updater $\mathcal{U}$, any messages $M, M' \in \Sigma^n$, the experiments $\mathbf{StrongTL}_{\mathcal{A},\mathcal{U},M}$ and $\mathbf{StrongTL}_{\mathcal{A},\mathcal{U},M'}$ are (computationally) indistinguishable.

3. Our Constructions

In this section, we present two constructions. As a warm-up, we first present a construction that is one-time secure to demonstrate the idea of achieving non-malleability, local decodability, and updatability simultaneously. Then in the next section, we show how to make the construction secure against continual attacks.

3.1. *Preliminary: Symmetric Encryption*

A symmetric encryption scheme consists of three PPT algorithms $(\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ such that:

- The key generation algorithm Gen takes as input a security parameter 1^k returns a key sk.
- The encryption algorithm $\mathsf{Encrypt}$ takes as input a key sk, and a message m. It returns a ciphertext $c \leftarrow \mathsf{Encrypt}_{\mathsf{sk}}(m)$.
- The decryption algorithm $\mathsf{Decrypt}$ takes as input a secret key sk, and a ciphertext c. It returns a message m or a distinguished symbol $\perp$. We write this as $m = \mathsf{Decrypt}_{\mathsf{sk}}(c)$

We require that for any m in the message space, it should hold that

$$\Pr[\mathsf{sk} \leftarrow \mathsf{Gen}(1^k);\ \mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{Encrypt}_{\mathsf{sk}}(m)) = m] = 1.$$

We next define semantical security and then the authenticity. In the following, we define a left-or-right encryption oracle $\mathsf{LR}_{\mathsf{sk},b}(\cdot,\cdot)$ with $b \in \{0, 1\}$ and $|m_0| = |m_1|$ as follows:

$$\mathsf{LR}_{\mathsf{sk},b}(m_0, m_1) \stackrel{\text{def}}{=} \mathsf{Encrypt}_{\mathsf{sk}}(m_b).$$

Definition 3.1. (*Semantical Security*) A symmetric encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is semantically secure if for any non-uniform PPT adversary $\mathcal{A}$, it holds that $|2 \cdot \mathbf{Adv}^{\text{priv}}_{\mathcal{E}}(\mathcal{A}) - 1| = \mathsf{negl}(k)$ where

$$\mathbf{Adv}^{\text{priv}}_{\mathcal{E}}(\mathcal{A}) = \Pr\left[\mathsf{sk} \leftarrow \mathsf{Gen}(1^k); b \leftarrow \{0,1\} : \mathcal{A}^{\mathsf{LR}_{\mathsf{sk},b}(\cdot,\cdot)}(1^k) = b\right].$$

Definition 3.2. (*Authenticity* [9,10,45]) A symmetric encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ has the property of authenticity if for any non-uniform PPT adversary $\mathcal{A}$, it holds that $\mathbf{Adv}^{\text{auth}}_{\mathcal{E}}(\mathcal{A}) = \mathsf{negl}(k)$ where

$$\mathbf{Adv}^{\text{auth}}_{\mathcal{E}}(\mathcal{A}) = \Pr[\mathsf{sk} \leftarrow \mathsf{Gen}(1^k), c^* \leftarrow \mathcal{A}^{\mathsf{Encrypt}_{\mathsf{sk}}(\cdot)} : c^* \notin \mathsf{Q} \wedge \mathsf{Decrypt}_{\mathsf{sk}}(c^*) \notin \perp]$$

where Q is the query history $\mathcal{A}$ made to the encryption oracle.

3.2. *A First Attempt: One-Time Security*

Construction Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a symmetric encryption scheme, $\mathsf{NMC} = (\text{ENC}, \text{DEC})$ be a coding scheme. Then we consider the following coding scheme:

- $\text{ENC}(M)$: on input $M = (m_1, m_2, \ldots, m_n)$, the algorithm first generates the encryption key $\mathsf{sk} \leftarrow \mathcal{E}.\mathsf{Gen}(1^k)$. Then it computes $c \leftarrow \mathsf{NMC}.\text{ENC}(\mathsf{sk})$, $e_i \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_i, i)$ for $i \in [n]$. The algorithm finally outputs a codeword $C = (c, e_1, e_2, \ldots, e_n)$.
- $\text{DEC}^C(i)$: on input $i \in [n]$, the algorithm reads the first block and the $(i+1)$-st block of the codeword to retrieve (c, e_i). Then it runs $\mathsf{sk} := \mathsf{NMC}.\text{DEC}(c)$. If the decoding algorithm outputs $\perp$, then it outputs $\perp$ and terminates. Else, it computes $(m_i, i^*) = \mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}}(e_i)$. If $i^* \neq i$, or the decryption fails, the algorithm outputs $\perp$. If all the above verifications pass, the algorithm outputs m_i.
- $\text{UPDATE}(i, m')$: on inputs an index $i \in [n]$, a block of message $m' \in \Sigma$, the algorithm runs $\text{DEC}^C(i)$ to retrieve (c, e_i) and (sk, m_i, i). If the decoding algorithm returns $\perp$, the algorithm writes $\perp$ to the first block and the $(i+1)$-st block. Otherwise, it computes a fresh encoding $c' \leftarrow \mathsf{NMC}.\text{ENC}(\mathsf{sk})$, and a fresh ciphertext $e'_i \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m', i)$. Then it writes back the first block and the $(i+1)$-st block with (c', e'_i).

To analyze the coding scheme, we make the following assumptions of the parameters in the underlying scheme for convenience:

1. The size of the encryption key is k (security parameter), i.e., $|\mathsf{sk}| = k$.
2. Let Σ be a set, and the encryption scheme supports messages of length $|\Sigma| + \log n$. The ciphertexts are in the space $\hat{\Sigma}$.
3. The length of $|\mathsf{NMC}.\text{ENC}(\mathsf{sk})|$ is less than $|\hat{\Sigma}|$.

Then clearly, the above coding scheme is an $(n, n+1, 2, 2)$-locally updatable and decodable code with respect to $\Sigma, \hat{\Sigma}$. The correctness of the scheme is obvious by inspection. The rate (ratio of the length of messages to that of codewords) of the coding scheme is $1 - o(1)$.

Theorem 3.3. *Assume $\mathcal{E}$ is a symmetric authenticated encryption scheme, and* NMC *is a non-malleable code against the tampering function class $\mathcal{F}$. Then the coding scheme presented above is one-time non-malleable against the tampering class*

$$\bar{\mathcal{F}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} f : \hat{\Sigma}^{n+1} \to \hat{\Sigma}^{n+1} \text{ and } |f| \le \text{poly}(k), \text{ such that :} \\ f = (f_1, f_2),\ f_1 : \hat{\Sigma}^{n+1} \to \hat{\Sigma},\ f_2 : \hat{\Sigma}^{n} \to \hat{\Sigma}^{n}, \\ \forall (x_2, \ldots, x_{n+1}) \in \hat{\Sigma}^n,\ f_1(\cdot, x_2, \ldots, x_{n+1}) \in \mathcal{F} \\ f(x_1, x_2, \ldots, x_{n+1}) = (f_1(x_1, x_2, \ldots, x_{n+1}),\ f_2(x_2, \ldots, x_{n+1})) \,. \end{array} \right\}.$$

We have presented the intuition in introduction. Before giving the detailed proof, we make the following remark.

Remark 3.4. The function class $\bar{\mathcal{F}}$ may look complex, yet the intuition is simple. The tampering function restricted in the first block (the underlying non-malleable code) falls into the class $\mathcal{F}$—this is captured by $f_1 \in \mathcal{F}$; on the other hand, we just require the function restricted in the rest of the blocks to be polynomial-sized—this is captured by $|f_2| \le |f| \le \text{poly}(k)$.

For our construction, it is inherent that the function f_2 cannot depend on x_1 arbitrarily. Suppose this is not the case, then f_2 can first decode the non-malleable code, encrypt the decoded value, and write the ciphertext into x_2, which breaks non-malleability. However, if the underlying coding scheme is non-malleable and also leakage resilient to $\mathcal{G}$, then we can allow f_2 to get additional information $g(x_1)$ for any $g \in \mathcal{G}$. Moreover, the above construction is one-time leakage resilient.

We present the above simpler version for clarity of exposition and give this remark that our construction actually achieves security against a broader class of tampering attacks.

Proof of Theorem 3.3. To show the theorem, for any function $f \in \bar{\mathcal{F}}$, we need to construct a PPT simulator $\mathcal{S}$ such that for any message blocks $M = (m_1, \ldots, m_n)$, we have $\mathbf{Tamper}^f_M \stackrel{c}{\approx} \mathbf{Ideal}_{\mathcal{S},M}$ as Definition 2.8. We describe the simulator as follows; here the PPT simulator $\mathcal{S}$ has oracle access to $f = (f_1, f_2) \in \bar{\mathcal{F}}$.

- $\mathcal{S}^{f(\cdot)}$ first runs $\mathsf{sk} \leftarrow \mathcal{E}.\mathsf{Gen}(1^k)$ and computes n encryptions of 0, i.e., $e_i \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(0)$ for $i \in [n]$.
- Let $f'_1(\cdot) \stackrel{\text{def}}{=} f_1(\cdot, e_1, e_2, \ldots, e_n)$, and let $\mathcal{S}'$ be the underlying simulator of the non-malleable code NMC with respect to the tampering function f'_1. Then $\mathcal{S}^{f(\cdot)}$ simulates $\mathcal{S}'^{f'_1(\cdot)}$ internally; here $\mathcal{S}$ uses the external oracle access to f to compute the responses for the queries made by $\mathcal{S}'$. At some point, $\mathcal{S}'$ returns an output $m' \in \Sigma \cup \{\mathsf{same}^*\}$.
- If $m' = \mathsf{same}^* \cup \{\mathsf{sk}\}$, then $\mathcal{S}$ computes $(e'_1, e'_2, \ldots, e'_n) \leftarrow f_2(e_1, e_2, \ldots, e_n)$. Let $\mathcal{I}$ be set of the indices where e' is not equal to e, i.e., $\mathcal{I} = \{i : e'_i \neq e_i\}$. Then $\mathcal{S}$ outputs $(\mathcal{I}, \vec{\epsilon})$, where $\vec{\epsilon}$ denotes the empty vector.
- Else (i.e., $m' \neq \mathsf{same}^* \cap m' \neq \mathsf{sk}$), $\mathcal{S}$ sets $\mathsf{sk}' := m'$, and computes $(e'_1, e'_2, \ldots, e'_n) \leftarrow f_2(e_1, e_2, \ldots, e_n)$, and sets $\vec{m}^* := (\mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}'}(e'_1), \ldots, \mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}'}(e'_n))$. Then $\mathcal{S}$ outputs $([n], \vec{m}^*)$.

To show $\mathbf{Tamper}^f_M \approx \mathbf{Ideal}_{\mathcal{S},M}$, we consider the following hybrids.

Hybrid H_0: This is exactly the experiment $\mathbf{Ideal}_{\mathcal{S},M}$.
Hybrid H_1: In this hybrid, we use a modified simulator, who is basically the same as $\mathcal{S}$, except in the first place the modified simulator generates $e_i \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_i, i)$ for $i \in [n]$.
Hybrid H_2: In this hybrid, we use another modified simulator, who is basically the same as the previous simulator, except this modified simulator obtains m' by running the real tampering experiment $f_1(\mathsf{NMC}.\text{ENC}(\mathsf{sk}), e_1, e_2, \ldots, e_n)$ and outputs same^* if the outcome is the original sk.

In the following claims, we are going to show that $\mathbf{Ideal}_{\mathcal{S},M} = H_0 \approx H_1 \approx H_2 \approx \mathbf{Tamper}^f_M$. Then the theorem follows directly from these claims.

Claim 3.5. *Suppose the encryption scheme $\mathcal{E}$ is semantically secure, then $H_0 \approx H_1$.*

Proof. Suppose an adversary can distinguish H_0 from H_1, then we can build a reduction to break semantic security of the encryption scheme as follows: the reduction gets input ciphertexts $e_1, \ldots, e_n$ that are either from $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_i, i)$ or $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(0)$ for $i \in [n]$. Then the reduction simulates the rest of the experiments. (Note $\mathcal{S}$ and $\mathcal{S}^*$ are identical expect for the ciphertexts.) Clearly, if the input ciphertexts come from $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(0)$, then the reduction identically simulates the experiment H_0, and otherwise H_1. Thus, if the adversary can distinguish the two experiments, then the reduction can break semantic security of the encryption. □

Claim 3.6. *Suppose* NMC *is a non-malleable code against $\mathcal{F}$, then $H_1 \approx H_2$.*

Proof. Since $f'_1 \in \mathcal{F}$, by the non-malleability of NMC, we have $\mathsf{NMC}.\mathsf{Tamper}^{f'_1}_{\mathsf{sk}} \approx \mathsf{NMC}.\mathbf{Ideal}_{\mathcal{S}',\mathsf{sk}}$. We note that the experiments H_2 and H_1 can be easily derived from $\mathsf{NMC}.\mathsf{Tamper}^{f'_1}_{\mathsf{sk}}$ and $\mathsf{NMC}.\mathbf{Ideal}_{\mathcal{S}',\mathsf{sk}}$, respectively, by setting m' as the output of either of the experiments (i.e., set $m' = \mathsf{NMC}.\mathsf{Tamper}^{f'_1}_{\mathsf{sk}}$ or $m' = \mathsf{NMC}.\mathbf{Ideal}_{\mathcal{S}',\mathsf{sk}}$). By the security of the coding scheme NMC, we know that $\mathsf{NMC}.\mathsf{Tamper}^{f'_1}_{\mathsf{sk}}$ and $\mathsf{NMC}.\mathbf{Ideal}_{\mathcal{S}',\mathsf{sk}}$ are indistinguishable. Therefore, H_1 and H_2 are indistinguishable as well. □

Claim 3.7. *Suppose the encryption scheme $\mathcal{E}$ has the property of authenticity, then $H_2 \approx \mathsf{Tamper}^f_M$.*

Proof. We first notice that suppose f'_1 modifies sk, then the two experiments execute identically. The only way that they differ is when sk remains unmodified, but there f_2 produces some "valid" e'_i which is not equal e_i. In this case H_2 would produce $\perp$, but Tamper^f_M will produce a meaningful message for that slot. We denote this as the event E. From the argument, we know that the statistical difference of the two experiments is bounded by $\Pr[E]$ as conditioned on $\neg E$, and these two experiments are identical.

Next we show that $\Pr[E] = \mathsf{negl}(k)$: suppose not, then we are going to build a reduction that breaks the authenticity property of the encryption scheme. The reduction queries the encryption oracle to obtain ciphertexts $e_1, \ldots, e_n$ and runs $(e'_1, \ldots, e'_n) \leftarrow f_2(e_1, \ldots, e_n)$. Then it randomly outputs an element in the set $\{e'_j : e'_j \neq e_j\}$. The

reduction succeeds as long as the event E happens, and it guesses a right modified ciphertext. This is with probability at least $\Pr[E]/n$, which is non-negligible. This reaches a contradiction.

Thus, we conclude that these two experiments are indistinguishable assuming the authenticity property of the encryption scheme. □

The proof of the theorem follows directly from these claims. □

3.3. *Achieving Security Against Continual Attacks*

As discussed in introduction, the above construction is not secure if continual tampering and leakage is allowed—the adversary can use a rewind attack to modify the underlying message to some old/related messages. We handle this challenge using a technique of Merkle tree, which preserves local properties of the above scheme. We present the construction in the following:

Definition 3.8. (*Merkle tree*) Let $h : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ be a hash function that maps two blocks of messages to one.[4] A Merkle tree $\mathsf{Tree}_h(M)$ takes input a message $M = (m_1, m_2, \ldots, m_n) \in \mathcal{X}^n$. Then it applies the hash on each pair (m_{2i-1}, m_{2i}), and resulting in $n/2$ blocks. Then again, it partitions the blocks into pairs and applies the hash on the pairs, which results in $n/4$ blocks. This is repeated $\log n$ times, resulting a binary tree with hash values, until one block remains. We call this value the root of Merkle tree denoted $\mathsf{Root}_h(M)$, and the internal nodes (including the root) as $\mathsf{Tree}_h(M)$. Here M can be viewed as leaves.

Theorem 3.9. *Assuming h is a collision-resistant hash function. Then for any message $M = (m_1, m_2, \ldots, m_n) \in \mathcal{X}^n$, any polynomial time adversary $\mathcal{A}$, $\Pr\left[(m'_i, p_i) \leftarrow \mathcal{A}(M, h) : m'_i \neq m_i,\ p_i \text{ is a consistent path with } \mathsf{Root}_h(M)\right] \leq \mathsf{negl}(k)$.*

Moreover, given a path p_i passing the leaf m_i, and a new value m'_i, there is an algorithm that computes $\mathsf{Root}_h(M')$ in time $\mathrm{poly}(\log n, k)$, where $M' = (m_1, \ldots, m_{i-1}, m'_i, m_{i+1}, \ldots, m_n)$.

Construction Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a symmetric encryption scheme, $\mathsf{NMC} = (\textsc{enc}, \textsc{dec})$ be a non-malleable code, H is a family of collision resistance hash functions. Then we consider the following coding scheme:

- $\textsc{enc}(M)$: on input $M = (m_1, m_2, \ldots, m_n)$, the algorithm first generates encryption key $\mathsf{sk} \leftarrow \mathcal{E}.\mathsf{Gen}(1^k)$ and $h \leftarrow H$. Then it computes $e_i \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m_i)$ for $i \in [n]$, and $T = \mathsf{Tree}_h(e_1, \ldots, e_n)$, $R = \mathsf{Root}_h(e_1, \ldots, e_n)$. Then it computes $c \leftarrow \mathsf{NMC}.\textsc{enc}(\mathsf{sk}, R, h)$. The algorithm finally outputs a codeword $C = (c, e_1, e_2, \ldots, e_n, T)$.
- $\textsc{dec}^C(i)$: on input $i \in [n]$, the algorithm reads the first block, the $(i+1)$-st block, and a path p in the tree (from the root to the leaf i), and it retrieves (c, e_i, p). Then it runs $(\mathsf{sk}, R, h) = \mathsf{NMC}.\textsc{dec}(c)$. If the decoding algorithm outputs $\perp$, or

[4] Here we assume $|\mathcal{X}|$ is greater than the security parameter.

the path is not consistent with the root R, then it outputs $\perp$ and terminates. Else, it computes $m_i = \mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}}(e_i)$. If the decryption fails, output $\perp$. If all the above verifications pass, the algorithm outputs m_i.

– UPDATE(i, m'): on inputs an index $i \in [n]$, a block of message $m' \in \Sigma$, the algorithm runs $\text{DEC}^C(i)$ to retrieve (c, e_i, p). Then the algorithm can derive $(\mathsf{sk}, R, h) = \mathsf{NMC}.\text{DEC}(c)$. If the decoding algorithm returns $\perp$, the update writes $\perp$ to the first block, which denotes failure. Otherwise, it computes a fresh ciphertext $e_i' \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m')$, a new path p' (that replaces e_i by e_i') and a new root R', which is consistent with the new leaf value e_i'. (Note that this can be done given only the old path p as Theorem 3.9.) Finally, it computes a fresh encoding $c' \leftarrow \mathsf{NMC}.\text{ENC}(\mathsf{sk}, R', h)$. Then it writes back the first block, the $(i+1)$-st block, and the new path blocks with (c', e_i', p').

To analyze the coding scheme, we make the following assumptions of the parameters in the underlying scheme for convenience:

1. The size of the encryption key is k (security parameter), i.e., $|\mathsf{sk}| = k$, and the length of the output of the hash function is k.
2. Let Σ be a set, and the encryption scheme supports messages of length $|\Sigma|$. The ciphertexts are in the space $\hat{\Sigma}$.
3. The length of $|\mathsf{NMC}.\text{ENC}(\mathsf{sk}, v)|$ is less than $|\hat{\Sigma}|$, where $|v| = k$.

Clearly, the above coding scheme is an $(n, 2n+1, O(\log n), O(\log n))$-locally updatable and decodable code with respect to $\Sigma, \hat{\Sigma}$. The correctness of the scheme is obvious by inspection. The rate (ratio of the length of messages to that of codewords) of the coding scheme is $1/2 - o(1)$.

Theorem 3.10. *Assume $\mathcal{E}$ is a semantically secure symmetric encryption scheme, and* NMC *is a non-malleable code against the tampering function class $\mathcal{F}$, and leakage resilient against the function class $\mathcal{G}$. Then the coding scheme presented above is non-malleable against continual attacks of the tampering class*

$$\bar{\mathcal{F}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} f : \hat{\Sigma}^{2n+1} \to \hat{\Sigma}^{2n+1} \text{ and } |f| \le \text{poly}(k), \text{ such that :} \\ f = (f_1, f_2),\ f_1 : \hat{\Sigma}^{2n+1} \to \hat{\Sigma},\ f_2 : \hat{\Sigma}^{2n} \to \hat{\Sigma}^{2n}, \\ \forall (x_2, \ldots, x_{2n+1}) \in \hat{\Sigma}^n,\ f_1(\cdot, x_2, \ldots, x_{2n+1}) \in \mathcal{F}, \\ f(x_1, x_2, \ldots, x_{2n+1}) = (f_1(x_1, x_2, \ldots, x_{2n+1}),\ f_2(x_2, \ldots, x_{2n+1})) . \end{array} \right\},$$

and is leakage resilient against the class

$$\bar{\mathcal{G}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} g : \hat{\Sigma}^{2n+1} \to \mathcal{Y} \text{ and } |g| \le \text{poly}(k), \text{ such that :} \\ g = (g_1, g_2),\ g_1 : \hat{\Sigma}^{2n+1} \to \mathcal{Y}',\ g_2 : \hat{\Sigma}^{2n} \to \hat{\Sigma}^{2n}, \\ \forall\ (x_2, \ldots, x_{2n+1}) \in \hat{\Sigma}^n, g_1(\cdot, x_2, \ldots, x_{2n+1}) \in \mathcal{G}. \end{array} \right\}.$$

The intuition of this construction can be found in introduction. Before giving the detailed proof, we make a remark.

Remark 3.11. Actually our construction is secure against a broader class of tampering functions. The f_2 part can depend on $g'(x_1)$ as long as the function $g'(\cdot)$ together with

the leakage function $g_1(\cdot, x_2, \ldots, x_{2n+1})$ belongs to $\mathcal{G}$. That is, the tampering function $f = (f_1, f_2, g')$ and the leakage function $g = (g_1, g_2)$ satisfy the constraint $g'(\cdot) \circ g_1(\cdot, x_2, \ldots, x_{2n+1}) \in \mathcal{G}$. (Here we use $\circ$ to denote concatenation.) For presentation clarity, we choose to describe the simpler but slightly smaller class of functions.

Proof of Theorem 3.10. To prove the theorem, for any adversary $\mathcal{A}$, we need to construct a simulator $\mathcal{S}$, such that for initial message $M \in \Sigma^n$, any updater $\mathcal{U}$, the experiment of continual attacks $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},M}$ is indistinguishable from the ideal experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$.

The simulator $\mathcal{S}$ first samples random coins for the updater $\mathcal{U}$, so its output just depends on its input given the random coins. Then $\mathcal{S}$ works as follows:

- Initially $\mathcal{S}$ samples $\mathsf{sk} \leftarrow \mathcal{E}.\mathsf{Gen}(1^k)$, $h \leftarrow H$, and then generates n encryptions of 0, i.e., $\vec{e}^{(1)} := (e_1, e_2, \ldots, e_n)$ where $e_i \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(0)$ for $i \in [n]$. Then $\mathcal{S}$ computes $T^{(1)} := \mathsf{Tree}_h(e_1, \ldots, e_n)$. Here let $R^{(1)}$ be the root of the tree. $\mathcal{S}$ keeps global variables: sk, h, a flag $\mathsf{flag} = 0$, and a string $C = \epsilon$ (empty string).
- At each round j, let $g^{(j)} \in \vec{\mathcal{G}}$, $f^{(j)} = (f_1^{(j)}, f_2^{(j)}) \in \vec{\mathcal{F}}$ be some leakage/tampering functions specified by the adversary. If the flag is 0, i.e., $\mathsf{flag} = 0$, then $\mathcal{S}$ does the following:
  - First, $\mathcal{S}$ sets $(e_1, e_2, \ldots, e_n) := \vec{e}^{(j)}$, $T := T^{(j)}$, and $R := R^{(j)}$. Then $\mathcal{S}$ defines $f_1'(\cdot) \stackrel{\text{def}}{=} f_1^{(j)}(\cdot, e_1, e_2, \ldots, e_n, T)$, and $g'(\cdot) \stackrel{\text{def}}{=} g^{(j)}(\cdot, e_1, e_2, \ldots, e_n, T)$. Let $\mathcal{S}'$ be the simulator of the underlying leakage-resilient non-malleable code NMC with respect to the tampering and leakage functions $f_1'(\cdot)$ and $g'(\cdot)$.
  - Then $\mathcal{S}$ computes $(m', \ell') \leftarrow \mathcal{S}'^{f_1'(\cdot), g'(\cdot)}$, and sets $\ell^{(j)} := \ell'$, and $(e_1', e_2', \ldots, e_n', T') := f_2^{(j)}(e_1, e_2, \ldots, e_n, T)$.
  - If $m' = \mathsf{same}^*$, $\mathcal{S}$ sets $\mathcal{I}^{(j)} = \{u : e_u' \neq e_u\}$, i.e., the indices where e' is not equal to e, and set $\vec{w}^{(j)} := \vec{\epsilon}$, the empty vector. $\mathcal{S}$ outputs $\{\ell^{(j)}, \mathcal{I}^{(j)}, \vec{w}^{(j)}\}$ for this round.
    Then upon receiving an index $i^{(j)} \in [n]$ from the updater, then $\mathcal{S}$ checks whether the path passing the leaf $e'_{i^{(j)}}$ in the Merkle tree T' is consistent with the root R, and does the following:
    - If the check fails, he sets $\mathsf{flag} := 1$, $C := (\perp, e_1', \ldots, e_n', T')$, and then exits the loop of this round.
    - Otherwise, he sets $\vec{e}^{(j+1)} := (e_1', e_2', \ldots, e_n')$ for all indices except $i^{(j)}$. He creates a fresh ciphertext $e \leftarrow \mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(0)$ and sets $\vec{e}^{(j+1)}[i^{(j)}] := e$ (simulating the update). He updates the path passing through the $i^{(j)}$-th leaf in T' and the root R, and set $T^{(j+1)} := T'$, $R^{(j+1)} := R$ (the updated ones).
  - Else if $m' \neq \mathsf{same}^*$, then $\mathcal{S}$ sets $\mathcal{I}^{(j)} := [n]$, and sets the flag to be 1, i.e., $\mathsf{flag} := 1$. He parses $m' = (\mathsf{sk}', h', R')$, and uses the key sk' to compute $\vec{w}^{(j)} = (\mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}'}(e_1'), \ldots, \mathcal{E}.\mathsf{Decrypt}_{\mathsf{sk}'}(e_n'))$. Then he outputs $\{\ell^{(j)}, \mathcal{I}^{(j)}, \vec{w}^{(j)}\}$ for this round.
    Then $\mathcal{S}$ computes $(i^{(j)}, m) \leftarrow \mathcal{U}(\vec{w}^{(j)})$ on his own. Let $C' = (\mathsf{NMC}.\textsc{enc}(\mathsf{sk}', h', R'), e_1', \ldots, e_n', T')$ be a codeword, and $\mathcal{S}$ runs $\textsc{update}^{C'}$

$(i^{(j)}, m)$. Let C^* be the resulting codeword, and $\mathcal{S}$ updates the global variable $C := C^*$.

– Else if $\mathsf{flag} = 1$, $\mathcal{S}$ simulates the real experiment faithfully:
  - $\mathcal{S}$ outputs $\ell^{(j)} = g^{(j)}(C)$, and computes $C' = f^{(j)}(C)$.
  - Set $\vec{w}^{(j)}[v] := \text{DEC}^{(C')}(v)$, i.e., running the real decoding algorithm. Then $\mathcal{S}$ outputs $\{\ell^{(j)}, \mathcal{I}^{(j)} = [n], \vec{w}^{(j)}\}$ for this round.
  - Then $\mathcal{S}$ computes $(i^{(j)}, m) \leftarrow \mathcal{U}(\vec{w}^{(j)})$ on his own and runs $\text{UPDATE}^{C'}(i^{(j)}, m)$. Let C^* be the resulting codeword after the update, and $\mathcal{S}$ updates the variable $C := C^*$.

To show $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M} \approx \mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$, we consider several intermediate hybrids.

Hybrid H_0: This is exactly the experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$.
Hybrid H_1: This experiment is the same as H_0 except the simulator does not generate sk of the encryption scheme. Whenever he needs to produce a ciphertext (only in the case when $\mathsf{flag} = 0$), the hybrid provides oracle access to the encryption algorithm $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(\cdot)$, where the experiment samples sk privately.

It is not hard to see that the experiment H_0 is identical to H_1. Then we define another hybrid:
Hybrid H_2: This experiment is the same as H_1 except; the encryption oracle does not give $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(0)$ to the simulator; instead, it gives encryptions of the real messages (in the first place, and in the update when $\mathsf{flag} = 0$), as in the real experiment.

Then we can establish the following claim.

Claim 3.12. *Suppose the encryption scheme is semantically secure, then H_1 is computationally indistinguishable from H_2.*

Proof. The proof is basically identical to the proof of Claim 3.5. Since the only difference between H_1 and H_2 is the encryption oracle's outputs (either $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(0)$ or $\mathcal{E}.\mathsf{Encrypt}_{\mathsf{sk}}(m)$ upon an input query m), if there is an adversary who can distinguish the two hybrids, then we can build a simple reduction that breaks the encryption scheme by a standard security proof argument. □

Next we consider the following hybrid.

Hybrid H_3: This experiment is the same as H_2 except, the simulator does not use the underlying $\mathcal{S}'$ of the non-malleable code to produce (m', ℓ') (in the case when $\mathsf{flag} = 0$). Let R be the current root of the Merkle tree, h be the hash function, sk be the secret key of the encryption oracle. In this experiment, the simulator generates an encoding of $\mathsf{NMC}.\text{ENC}(\mathsf{sk}, h, R)$ and then applies the tampering and leakage function faithfully as the real experiment $\mathbf{TamperLeak}^{f,g}$. If the outcome is still (sk, h, R), then the simulator treats this as same^*. Otherwise, it uses the decoded value to proceed. Then the rest follows exactly as H_2.

Then we can establish the following claim:

Claim 3.13. *Suppose the underlying coding scheme* NMC *is non-malleable and leakage resilience against $\mathcal{F}$ and $\mathcal{G}$, then H_2 is computationally indistinguishable from H_3.*

Proof. We can show this by considering the following sub-hybrids: $H_{2,j}$: in the first j rounds, the simulator generates (m', ℓ') according to the experiment **TamperLeak** and in the rest $\mathcal{S}'$. By the property of the coding scheme, we can show each adjacent sub-hybrid is computationally indistinguishable. Note that the simulator refreshes the encoding of (sk, h, R) at each round, so we can apply the hybrid argument. From the description, we have $H_2 = H_{2,0}$ and $H_{2,t} = H_3$, where t is the total number of rounds.

More formally, suppose $H_{2,j}$ and $H_{2,j+1}$ are distinguishable for some $j \in [t-1]$, then we construct a reduction that distinguishes the experiment $\mathbf{TamperLeak}_m^{f,g}$ from $\mathbf{Ideal}_{\mathcal{S}',m}$ for some functions $f \in \mathcal{F}$, $g \in \mathcal{G}$ and some message m. This is a contradiction to the security property of the underlying coding scheme. The message m can be set to (sk, h, R), and f, g be the functions in the j-th round output by the adversary. The reduction first simulates the experiment up to the j-th round. At this moment, $H_{2,j}$ and $H_{2,j+1}$ are identical. Then the reduction receives an input (m', ℓ') from either of the two experiments. The reduction uses these values for the round $j+1$. Note that it is sufficient for the reduction to finish the round $j+1$ by knowing (m', ℓ'). Finally the reduction continues simulating the rest of the experiment. It is clear that if (m', ℓ') is from $\mathbf{TamperLeak}_m^{f,g}$, then the reduction simulates $H_{2,j+1}$, yet otherwise $H_{2,j}$. Thus, if the two adjacent hybrids are distinguishable, the reduction can break the leakage-resilient non-malleable code of the underlying scheme. The completes the proof of the claim. □

Finally we want to show the following claim:

Claim 3.14. *Suppose the hash function comes from a collision-resilient hash family, then H_3 is computationally indistinguishable from* $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M}$.

Proof. We observe that the only difference between H_3 and $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M}$ is the generation of $\vec{m}^{(j)}$ at each round (when the flag is 0). In the experiment $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M}$, $\vec{m}^{(j)}$ is generated by honestly decoding the codeword at each position, i.e., $\left(\mathrm{DEC}^{f(C^{(j)})}(1), \ldots, \mathrm{DEC}^{f(C^{(j)})}(n)\right)$. In H_3, $\vec{m}^{(j)}$ is generated by first computing $(m', e'_1, \ldots, e'_n, T') := f(C^{(j)})$. In the case where $m' \neq \mathsf{same}^*$, the two experiments are identical. In the case where $m' = \mathsf{same}^*$, H_3 sets $\vec{m}^{(j)}[v] = \perp$ if $e'_v \neq e_v$. The only situation that these two hybrids deviate is when $e'_v \neq e_v$, but there is another consistent path in T' with the root R. For this situation, $\mathrm{DEC}(v) \neq \perp$ in **CTamperLeak**, but H_3 will set $\vec{m}[v] := \perp$. However, we claim this event can happen with a negligible probability, or otherwise we can break the security of the Merkle tree (Theorem 3.9) by simulating the hybrid H_3. This completes the proof of the claim. □

Putting everything together, we show that $\mathbf{CTamperLeak}_{\mathcal{A},\mathcal{U},M} \approx \mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$. □

Remark 3.15. Our one-time and continual constructions in Sects. 3.2 and 3.3, respectively, achieve the notion of *strong* non-malleability if the underlying non-malleable

code is itself a strong non-malleable code. In the continual construction (see Sect. 3.3), in order to prove *strong* non-malleability we must show that the adversary's view is indistinguishable when it receives codeword $C = (c, e_1, e_2, \ldots, e_n, T)$, where $e_1, \ldots, e_n$ are encryptions of $m_1, \ldots, m_n$ and when it receives codeword $C' = (c, e'_1, e'_2, \ldots, e'_n, T)$, where $e'_1, \ldots, e'_n$ are encryptions of $m'_1, \ldots, m'_n$. This can be shown using a sequence of hybrids, similar to the one used in Sect. 3.3. The main difference is that in the first hybrid, we explicitly set $c \leftarrow \mathsf{NMC}.\textsc{enc}(0, 0, 0)$ (i.e., generate a non-malleable codeword encoding message $(0, 0, 0)$ as opposed to (sk, R, h) in the real construction) and use *strong* non-malleability of the underlying code to argue indistinguishability. The remaining hybrids follow similarly to those of Sect. 3.3.

3.4. *Instantiations*

In this section, we describe several constructions of non-malleable codes against different classes of tampering/leakage functions. To our knowledge, we can use the explicit constructions (of the non-malleable codes) in the works [1,3–5,7,17,29,30,32,50].

First we overview different classes of tampering/leakage function allowed for these results: the constructions of Dziembowski et al. [29] work for bit-wise tampering functions and split-state functions in the random oracle model. The construction of Choi et al. [17] works for small block tampering functions. The construction of Liu and Lysyanskaya [50] achieves both tamper and leakage resilience against split-state functions in the common reference string (CRS) model. The construction of Dziembowski et al. [27] achieves information-theoretic security against split-state tampering functions, but their scheme can only support encoding for bits, so it cannot be used in our construction. The subsequent construction by Aggarwal et al. [3] achieves information-theoretic security against split-state tampering without CRS. Recently, Aggarwal et al. [4] presented information-theoretic non-malleable codes that achieve both tamper and leakage resilience against split-state functions and do not require a common reference string. Subsequently, Aggarwal et al. [1] achieved optimal leakage rate, but only in the computational setting, and Aggarwal et al. [5] further strengthened these results by extending to the continual setting. The construction by Faust et al. [32] is non-malleable against small-sized tampering functions. Another construction by Faust et al. [30] achieves both tamper and leakage resilience in the split-state model with CRS. The construction of Aggarwal et al. [7] is non-malleable against permutation functions.

We also remark that there are other non-explicit constructions: Cheraghchi and Guruswami [16] showed the relation non-malleable codes and non-malleable two-source extractors (but constructing a non-malleable two-source extractor is still open), and in another work Cheraghchi and Guruswami [15] showed the existence of high rate non-malleable codes in the split-state model but did not give an explicit (efficient) construction.

Finally, we give a concrete example of what the resulting class looks like using the construction of Liu and Lysyanskaya [50] as the building block—recall that their construction achieves both tamper and leakage resilience for split-state functions (essentially the same class of leakage/tampering would be achieved by plugging in any of the split-state constructions). Our construction has the form $(\mathsf{NMC}.\textsc{enc}(\mathsf{sk}, h, T), \mathsf{Encrypt}(m_1), \ldots, \mathsf{Encrypt}(m_n), T)$. So the overall leakage function g restricted in the first block (i.e., g_1)

can be a (poly-sized) length-bounded split-state function; g, on the other hand, can leak all the other parts. For the tampering, the overall tampering function f restricted in the first block (i.e., f_1) can be any (poly-sized) split-state function. On the other hand f restricted in the rest (i.e., f_2) can be just any poly-sized function. We also remark that f_2 can depend on a split-state leakage on the first part, say g_1, as we discussed in the previous remark above.

4. Tamper and Leakage-Resilient RAM

In this section, we first introduce the notations of the random access machine (RAM) model of computation in the presence of tampering and leakage attacks in Sect. 4.1. Then we define the security of tamper and leakage-resilient RAM model of computation in Sect. 4.2, recall the building block oblivious RAM (ORAM) in Sect. 4.3, and then give a construction in Sect. 4.4 and the security analysis in Sect. 4.5.

4.1. *Random Access Machines*

We consider RAM programs to be interactive stateful systems $\langle \Pi, \mathsf{state}, D \rangle$, where Π denotes a next instruction function, state the current state stored in registers, and D the content of memory. Upon state and an input value d, the next instruction function outputs the next instruction I and an updated state state'. The initial state of the RAM machine, state, is set to $(\mathsf{start}, *)$. For simplicity we often denote RAM program as $\langle \Pi, D \rangle$. We consider four ways of interacting with the system:

- $\mathsf{Execute}(x)$: A user can provide the system with $\mathsf{Execute}(x)$ queries, for $x \in \{0,1\}^u$, where u is the input length. Upon receiving such query, the system computes $(y, t, D') \leftarrow \langle \Pi, D \rangle(x)$, updates the state of the system to $D := D'$ and outputs (y, t), where y denotes the output of the computation and t denotes the time (or number of executed instructions). By $\mathsf{Execute}_1(x)$ we denote the first coordinate of the output of $\mathsf{Execute}(x)$.
- $\mathsf{doNext}(x)$: A user can provide the system with $\mathsf{doNext}(x)$ queries, for $x \in \{0,1\}^u$. Upon receiving such query, if $\mathsf{state} = (\mathsf{start}, *)$, set $\mathsf{state} := (\mathsf{start}, x)$, and $d := 0^r$; here $\rho = |\mathsf{state}|$ and $r = |d|$. The system does the following until termination:
  1. Compute $(I, \mathsf{state}') = \Pi(\mathsf{state}, d)$. Set $\mathsf{state} := \mathsf{state}'$.
  2. If $I = (\mathsf{wait})$, then set $\mathsf{state} := 0^\rho$, $d := 0^r$ and terminate.
  3. If $I = (\mathsf{stop}, z)$, then set $\mathsf{state} := (\mathsf{start}, *)$, $d := 0^r$ and terminate with output z.
  4. If $I = (\mathsf{write}, v, d')$, then set $D[v] := d'$.
  5. If $I = (\mathsf{read}, v, \perp)$, then set $d := D[v]$.

  Let $I_1, \ldots, I_\ell$ be the instructions executed by $\mathsf{doNext}(x)$. All memory addresses of executed instructions are returned to the user. Specifically, for instructions I_j of the form $(\mathsf{read}, v, \perp)$ or (write, v, d'), v is returned.

- $\mathsf{Tamper}(f)$: We also consider tampering attacks against the system, modeled by $\mathsf{Tamper}(f)$ commands, for functions f. Upon receiving such command, the system sets $D := f(D)$.
- $\mathsf{Leak}(g)$: We finally consider leakage attacks against the system, modeled by $\mathsf{Leak}(g)$ commands, for functions g. Upon receiving such command, the value of $g(D)$ is returned to the user.

Remark 4.1. A $\mathsf{doNext}(x)$ instruction groups together instructions performed by the CPU in a single clock cycle. Intuitively, a (wait) instruction indicates that a clock cycle has ended and the CPU waits for the adversary to increment the clock. In contrast, a (stop, z) instruction indicates that the entire execution has concluded with output z. In this case, the internal state is set back to the start state.

We require that each $\mathsf{doNext}(x)$ operation performs exactly $\ell = \ell(k) = \mathrm{poly}(k)$ instructions $I_1, \ldots, I_\ell$ where the final instruction is of the form $I_\ell = (\mathsf{stop}, \cdot)$ or $I_\ell = (\mathsf{wait})$. For fixed $\ell_1 = \ell_1(k), \ell_2 = \ell_2(k)$ such that $\ell_1 + \ell_2 = \ell - 1$, we have that the first ℓ_1 instructions are of the form $I_\ell = (\mathsf{read}, \cdot, \perp)$ and the next ℓ_2 instructions are of the form $I_\ell = (\mathsf{write}, v, d')$. We assume that ℓ, ℓ_1, ℓ_2 are implementation-specific and public. The limitations on space are meant to model the fact that the CPU has a limited number of registers and that no persistent state is kept by the CPU between clock cycles.

Remark 4.2. We note that $\mathsf{Execute}(x)$ instructions are used by the ideal world adversary—who learns only the input-output behavior of the RAM machine and the run time—as well as by the real-world adversary. The real-world adversary may also use the more fine-grained $\mathsf{doNext}(x)$ instruction. We note that given access to the $\mathsf{doNext}(x)$ instruction, the behavior of the $\mathsf{Execute}(x)$ instruction may be simulated.

4.1.1. *Dealing with Leakage and Tampering on Instructions I*

We note that our model does not explicitly allow for leakage and tampering on instructions I. E.g., when an instruction $I = (\mathsf{write}, v, d')$ is executed, we do not directly allow tampering with the values v, d' or leakage on d'. (Note that v is entirely leaked to the adversary.) Nevertheless, as discussed in introduction, since we allow full leakage on the addresses, the adversary can in some instances use the tampering and leakage attacks on the memory to simulate attacks on the instructions. We elaborate in the following:

Leakage on an instruction I Since (write, v) or (read, v) is entirely leaked to the adversary, we need only deal with leakage on d'. In this case, an adversary leaking on d' can be simulated in our model by an adversary who leaks the contents of memory location v in the following round.

Tampering with d' **in an instruction** I **of the form** $I = (\mathsf{read}, v, d')$ In this case, adversarial tampering will have no effect.

Tampering with d' **in an instruction** I **of the form** $I = (\mathsf{write}, v, d')$ In this case, adversarial tampering with d' can be simulated in our model by an adversary who tampers with the contents of memory location v in the following round.

Tampering with v **in an instruction** I **of the form** $I = (\mathsf{read}, v, d')$ Such tampering is not straightforwardly captured by our model. We can change our model so that each round has two stages: in the first stage, the adversary is given all instructions

$I_1, \ldots, I_\ell$ and then the adversary may pre-emptively leak and tamper before the instructions are completed. Security of our construction still holds in this slightly modified setting. To simulate tampering on v, an adversary can now leak the contents of memory location v, apply a tampering function which will copy the contents of the tampered location $\widetilde{v}$ to location v, allow the system to read the memory location, and then write back the old contents of memory to v in the next round.

Tampering with d' in an instruction I of the form $I = (\mathsf{write}, v, d')$ Such tampering is not straightforwardly captured by our model. We can change our model so that each round has two stages: in the first stage, the adversary is given all instructions $I_1, \ldots, I_\ell$ and then the adversary may pre-emptively leak and tamper before the instructions are completed. Security of our construction still holds in this slightly modified setting. To simulate tampering on v, an adversary can now leak the contents of memory location v, apply a tampering function which will place d' in the tampered location $\widetilde{v}$, allow the system to write to memory location v, and then write back the old contents of memory to v in the next round.

Tampering with read or write in an instruction I Our model does not handle this type of tampering.

4.2. *Tamper and Leakage-Resilient (TLR) RAM*

A tamper and leakage-resilient (TLR) RAM compiler consists of two algorithms $(\mathsf{CompMem}, \mathsf{CompNext})$, which transform a RAM program $\langle \Pi, D \rangle$ into another program $\langle \widehat{\Pi}, \widehat{D} \rangle$ as follows: on input database D, $\mathsf{CompMem}$ initializes the memory and internal state of the compiled machine and generates the transformed database $\widehat{D}$; on input next instruction function Π, $\mathsf{CompNext}$ generates the next instruction function of the compiled machine.

Definition 4.3. A TLR compiler $(\mathsf{CompMem}, \mathsf{CompNext})$ is tamper and leakage simulatable w.r.t. function families $\mathcal{F}, \mathcal{G}$, if for every RAM next instruction function Π, and for any PPT (non-uniform) adversary $\mathcal{A}$ there exists a PPT (non-uniform) simulator $\mathcal{S}$ such that for any initial database $D \in \{0,1\}^{\mathrm{poly}(k)}$ we have

$$\mathbf{TamperExec}(\mathcal{A}, \mathcal{F}, \mathcal{G}, \langle \mathsf{CompNext}(\Pi), \mathsf{CompMem}(D) \rangle) \approx \mathbf{IdealExec}(\mathcal{S}, \langle \Pi, D \rangle)$$

where **TamperExec** and **IdealExec** are defined as follows:

- **TamperExec**$(\mathcal{A}, \mathcal{F}, \mathcal{G}, \langle \mathsf{CompNext}(\Pi), \mathsf{CompMem}(D) \rangle)$: The adversary $\mathcal{A}$ interacts with the system $\langle \mathsf{CompNext}(\Pi), \mathsf{CompMem}(D) \rangle$ for arbitrarily many rounds of interactions where in each round:
  1. The adversary can "tamper" by executing a $\mathsf{Tamper}(f)$ command against the system, for some $f \in \mathcal{F}$.
  2. The adversary can "leak" by executing a $\mathsf{Leak}(g)$ command against the system, and receiving $g(D)$ in return.
  3. The adversary requests a $\mathsf{doNext}(x)$ command to be executed by the system. Let $I_1, \ldots, I_\ell$ be the instructions executed by $\mathsf{doNext}(x)$. If I_ℓ is of the form

(stop, z), then output z is returned to the adversary. Moreover, all memory addresses corresponding to instructions $I_1, \ldots, I_{\ell-1}$ are returned to the adversary.

The output of the game consists of the output of the adversary $\mathcal{A}$ at the end of the interaction, along with (1) all input–output pairs $(x_1, y_1), (x_2, y_2), \ldots$, (2) all responses to leakage queries $\ell_1, \ell_2, \ldots$, (3) all outputs of $\mathsf{doNext}(x_1), \mathsf{doNext}(x_2), \ldots$.

– **IdealExec**$(\mathcal{S}, \langle \Pi, D \rangle)$: The simulator interacts with the system $\langle \Pi, D \rangle$ for arbitrarily many rounds of interaction where, in each round, it runs an $\mathsf{Execute}(x)$ query for some $x \in \{0,1\}^u$ and receives output (y, t). The output of the game consists of the output of the simulator $\mathcal{S}$ at the end of the interaction, along with all of the execute-query inputs and outputs.

For simplicity of exposition, we assume henceforth that the next instruction function Π to be compiled is the universal RAM next instruction function. In other words, we assume that the program to be executed is stored in the initial database D.

4.3. *Preliminary: Oblivious RAM (ORAM)*

An ORAM compiler ORAM consists of two algorithms ($\mathsf{oCompMem}$, $\mathsf{oCompNext}$), which transform a RAM program $\langle \Pi, D \rangle$ into another program $\langle \widetilde{\Pi}, \widetilde{D} \rangle$ as follows: on input database D, $\mathsf{CompMem}$ initializes the memory and internal state of the compiled machine and generates the transformed database $\widetilde{D}$; on input next instruction function Π, $\mathsf{CompNext}$ generates the next instruction function of the compiled machine, $\widetilde{\Pi}$.

Correctness We require the following correctness property: for every choice of security parameter k, every initial database D, and every sequence of inputs $x_1, \ldots, x_p$, where $p = p(k)$ is polynomial in k, we have that with probability $1 - \mathsf{negl}(k)$ over the coins of $\mathsf{oCompMem}$,

$$\left(\mathsf{Execute}_1(x_1), \ldots, \mathsf{Execute}_1(x_p)\right) = \left(\widetilde{\mathsf{Execute}}_1(x_1), \ldots, \widetilde{\mathsf{Execute}}_1(x_p)\right),$$

where $\mathsf{Execute}_1(x)$ denotes the first coordinate of the output of $\mathsf{Execute}(x)$ w.r.t. $\langle \Pi, D \rangle$ and $\widetilde{\mathsf{Execute}}_1(x)$ denotes the first coordinate of the output of $\mathsf{Execute}(x)$ w.r.t. $\langle \mathsf{oCompNext}(\Pi), \mathsf{oCompMem}(D) \rangle$.

Security Let $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$ be an ORAM complier and consider the following experiment:
Experiment $\mathbf{Expt}^{\mathrm{oram}}_{\mathcal{A}}(k, b)$:

1. The adversary $\mathcal{A}$ selects two initial databases D_0, D_1.
2. Set initial contents of memory of the RAM machine to $\widetilde{D} := \mathsf{oCompMem}(D_b)$. Set the initial state of the RAM machine to $\mathsf{state} := (\mathsf{start}, *)$.
3. The adversary $\mathcal{A}$ and the challenger participate in the following procedure for an arbitrary number of rounds:
   - For $x \in \{0,1\}^u$, $\mathcal{A}$ submits a $\mathsf{doNext}(x)$ query.
   - Execute the $\mathsf{doNext}(x)$ query w.r.t. $\langle \mathsf{oCompNext}(\Pi), \widetilde{D} \rangle$ and update the state of the system. Let $I_1, \ldots, I_\ell$ be the instructions executed by the RAM machine.

For each $j \in [\ell]$, if I_j is of the form $(\cdot, v_j, \cdot)$, for some v_j, output v_j to $\mathcal{A}$. Otherwise, output $v_j = \perp$. Let $\overline{v} = v_1, \ldots, v_\ell$ be the output obtained by $\mathcal{A}$ in the current round.

4. Finally, the adversary outputs a guess $b' \in \{0, 1\}$. The experiment evaluates to 1 iff $b' = b$.

Definition 4.4. An ORAM construction $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$ is *access-pattern hiding* if for every PPT adversary $\mathcal{A}$, the following probability, taken over the randomness of the experiment and $b \in \{0, 1\}$, is negligible:

$$\left| \Pr[\mathbf{Expt}^{\text{oram}}_{\mathcal{A}}(k, b) = 1] - \frac{1}{2} \right|.$$

4.4. *TLR-RAM Construction*

Here we first give a high-level description of our construction. More detailed construction and a theorem statement follow. The security proof will be given in the next section.

High-level Description of Construction Let D be the initial database, and let $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$ be an ORAM compiler. Let $\mathsf{NMCode} = (\text{ENC}, \text{DEC}, \text{UPDATE})$ be a locally decodable and updatable code. We present the following construction $\mathsf{TLR\text{-}RAM} = (\mathsf{CompMem}, \mathsf{CompNext})$ of a tamper and leakage-resilient RAM compiler. In order to make our presentation more intuitive, instead of specifying the next message function $\mathsf{CompNext}(\Pi)$, we specify the pseudocode for the $\mathsf{doNext}(x)$ instruction of the compiled machine. We note that $\mathsf{CompNext}(\Pi)$ is implicitly defined by this description.

TLR-RAM takes as input an initial database D and a next instruction function Π and does the following:

- CompMem: On input security parameter k and initial database D, CompMem does:
  - Compute $\widetilde{D} \leftarrow \mathsf{oCompMem}(D)$, and output $\widehat{D} \leftarrow \text{ENC}(\widetilde{D})$.
  - Initialize the ORAM state $\mathsf{state}_{\mathsf{ORAM}} := (\mathsf{start}, *)$ and $d_{\mathsf{ORAM}} := 0^r$, where $r = |d_{\mathsf{ORAM}}|$.
- $\mathsf{doNext}(x)$: On input x, do the following until termination:
  1. If $d_{\mathsf{ORAM}} = \perp$ then abort.
  2. Compute $(I, \mathsf{state}'_{\mathsf{ORAM}}) \leftarrow \mathsf{oCompNext}(\Pi)(\mathsf{state}_{\mathsf{ORAM}}, d_{\mathsf{ORAM}})$. Set $\mathsf{state}_{\mathsf{ORAM}} := \mathsf{state}'_{\mathsf{ORAM}}$.
  3. If $I = (\mathsf{wait})$ then set $\mathsf{state}_{\mathsf{ORAM}} := 0^\rho$ and $d_{\mathsf{ORAM}} := 0^r$ and terminate. Here $\rho = |\mathsf{state}_{\mathsf{ORAM}}|$ and $r = |d_{\mathsf{ORAM}}|$.
  4. If $I = (\mathsf{stop}, z)$ then set $\mathsf{state}_{\mathsf{ORAM}} := (\mathsf{start}, *)$, $d := 0^r$ and terminate with output z.
  5. If $I = (\mathsf{write}, v, d')$ then run $\text{UPDATE}^{\widehat{D}}(v, d')$.
  6. If $I = (\mathsf{read}, v, \perp)$ then set $d_{\mathsf{ORAM}} := \text{DEC}^{\widehat{D}}(v)$.

Detailed Description of Construction Let $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$ be an ORAM compiler and let $\mathsf{NMCode} = (\text{ENC}, \text{DEC}, \text{UPDATE})$ be a locally decodable and updatable code. We view DEC and UPDATE as RAM machines and denote by Π_{DEC}, Π_{UPDATE} the corresponding next message functions. We present the following tamper and leakage-resilient RAM compiler $\mathsf{TLR\text{-}RAM} = (\mathsf{CompMem}, \mathsf{CompNext})$. Here, the parameters $r = r(k)$, $u = u(k)$, $\rho = \rho(k)$ are polynomials in the security parameter k that are implementation-dependent. The complier TLR-RAM takes as input an initial database D and a next instruction function Π and does the following:

CompMem: On input security parameter k and initial database D, CompMem does the following:

- Run oCompMem to compute $\widetilde{D} \leftarrow \mathsf{oCompMem}(D)$, and to initialize $\mathsf{state}_{\mathsf{ORAM}} := (\mathsf{start}, *)$, and $d_{\mathsf{ORAM}} := 0^r$.
- Output $\widehat{D} \leftarrow \text{ENC}(\widetilde{D})$.
- Initialize $\mathsf{state} \stackrel{\text{def}}{=} \mathsf{state}_{\mathsf{ORAM}}||\mathsf{state}_{\mathsf{code}}||\mathsf{mode} := (\mathsf{start}, *)||(\mathsf{start}, *)||\perp$ and $d \stackrel{\text{def}}{=} d_{\mathsf{code}}||d_{\mathsf{ORAM}} := 0^r||0^r$.

CompNext: On input next instruction function Π, let $\widetilde{\Pi} = \mathsf{oCompNext}(\Pi)$ be the next instruction function of the ORAM compiled machine. $\mathsf{CompNext}(\Pi)$ is the next instruction function of the TLR-RAM compiled machine. It takes as input (state, d) and does the following:

- Parse $\mathsf{state} = \mathsf{state}_{\mathsf{ORAM}}||\mathsf{state}_{\mathsf{code}}||\mathsf{mode}$. Here $\mathsf{mode} \in \{\text{UP}, \text{DEC}, \perp\}$
- If $d_{\mathsf{ORAM}} = \perp$ then abort.
- If $\mathsf{state}_{\mathsf{code}} = (\mathsf{start}, *)$: Compute $(I_{\mathsf{ORAM}}, \mathsf{state}'_{\mathsf{ORAM}}) := \widetilde{\Pi}(\mathsf{state}_{\mathsf{ORAM}}, d_{\mathsf{ORAM}})$.
  1. If I_{ORAM} is of the form $I_{\mathsf{ORAM}} = (\mathsf{wait})$ then set $I := (\mathsf{wait})$.
     Set $\mathsf{state} := \mathsf{state}'_{\mathsf{ORAM}}||\mathsf{state}_{\mathsf{code}}||\mathsf{mode}$.
     Output (I, state).
  2. If I_{ORAM} is of the form (stop, z) then set $I := (\mathsf{stop}, z)$.
     Set $\mathsf{state} := \mathsf{state}'_{\mathsf{ORAM}}||\mathsf{state}_{\mathsf{code}}||\mathsf{mode}$.
     Output (I, state).
  3. If I_{ORAM} is of the form (write, v, d') then set $\mathsf{state}_{\mathsf{code}} := (\mathsf{start}, v, d')$.
     Set $I := (\mathsf{read}, 0, \perp)$ where $(\mathsf{read}, 0, \perp)$ denotes a dummy read.
     Set $\mathsf{state} := \mathsf{state}_{\mathsf{ORAM}}||\mathsf{state}'_{\mathsf{code}}||\text{UP}$.
     Output (I, state).
  4. If I_{ORAM} is of the form $(\mathsf{read}, v, \perp)$ then set $\mathsf{state}_{\mathsf{code}} := (\mathsf{start}, v)$.
     Set $I := (\mathsf{read}, 0, \perp)$ where $(\mathsf{read}, 0, \perp)$ denotes a dummy read.
     Set $\mathsf{state} := \mathsf{state}_{\mathsf{ORAM}}||\mathsf{state}'_{\mathsf{code}}||\text{DEC}$.
     Output (I, state).
- Otherwise if $\mathsf{state}_{\mathsf{code}} \neq (\mathsf{start}, *)$:
  If $\mathsf{mode} = \text{UP}$, compute $(I_{\mathsf{code}}, \mathsf{state}'_{\mathsf{code}}) := \Pi_{\text{UPDATE}}(\mathsf{state}_{\mathsf{code}}, d_{\mathsf{code}})$.
  If $\mathsf{mode} = \text{DEC}$, compute $(I_{\mathsf{code}}, \mathsf{state}'_{\mathsf{code}}) := \Pi_{\text{DEC}}(\mathsf{state}_{\mathsf{code}}, d_{\mathsf{code}})$.
  1. If I_{code} is of the form (stop, z) then set $I := (\mathsf{read}, 0, \perp)$, where $(\mathsf{read}, 0, \perp)$ denotes a dummy read.

Set $d_{\mathsf{ORAM}} := z$, set $\mathsf{state}'_{\mathsf{code}} := (\mathsf{start}, *)$, set $\mathsf{state} := \mathsf{state}'_{\mathsf{ORAM}} || \mathsf{state}'_{\mathsf{code}} || \perp$.
Output (I, state).

2. If I_{code} is of the form $(\mathsf{read}, \hat{v}, \perp)$, set $I := I_{\mathsf{code}}$.
Set $\mathsf{state} := \mathsf{state}'_{\mathsf{ORAM}} || \mathsf{state}'_{\mathsf{code}} || \mathrm{DEC}$.
Output (I, state).
3. If I_{code} is of the form $(\mathsf{write}, \hat{v}, \hat{d}')$, set $I := I_{\mathsf{code}}$.
Set $\mathsf{state} := \mathsf{state}'_{\mathsf{ORAM}} || \mathsf{state}'_{\mathsf{code}} || \mathrm{UP}$.
Output (I, state).
Upon execution of I, d_{code} will be set to $\widehat{D}[\hat{v}]$.

We are now ready to present the main theorem of this section:

Theorem 4.5. *Assume* $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$ *is an ORAM compiler which is access-pattern hiding and assume* $\mathsf{NMCode} = (\mathrm{ENC}, \mathrm{DEC}, \mathrm{UPDATE})$ *is a locally decodable and updatable code which is continual non-malleable against* $\mathcal{F}$ *and leakage resilient against* $\mathcal{G}$. *Then* $\mathsf{TLR\text{-}RAM} = (\mathsf{CompMem}, \mathsf{CompNext})$ *presented above is tamper and leakage simulatable w.r.t. function families* $\mathcal{F}, \mathcal{G}$.

4.5. *Security Analysis*

In this section we prove Theorem 4.5. We begin by defining the simulator $\mathcal{S}$. Let $\mathcal{S}_{\mathsf{code}}$ be the simulator guaranteed by the security of $\mathsf{NMCode} = (\mathrm{ENC}, \mathrm{DEC}, \mathrm{UPDATE})$.

For simplicity of exposition, we assume that for every x, given the runtime t of $\mathsf{Execute}(x)$ with respect to $\langle \Pi, D \rangle$, the runtime of $\mathsf{Execute}(x)$ with respect to $\langle \mathsf{oCompNext}(\Pi), \mathsf{oCompMem}(D) \rangle$ is equal to $p(t)$, $p(\cdot)$ is a fixed polynomial known to the simulator. This is indeed the case for the instantiation of our compiler with known underlying building blocks.

Simulator $\mathcal{S}$

Setup: On input security parameter k, $\mathcal{S}$ does the following:

- Choose a dummy database D_0, compute $\widetilde{D} \leftarrow \mathsf{oCompMem}(D_0)$. Initialize $\mathsf{state}_{\mathsf{ORAM}} := (\mathsf{start}, *)$, $d_{\mathsf{ORAM}} = 0^r$.
- Instantiate the adversary $\mathcal{A}$ and the NMCode simulator $\mathcal{S}_{\mathsf{code}}$.
- Initialize output variable $\mathsf{out} = \perp$ and counter $c = 0$.

Adversarial query $(g, f, \mathsf{doNext}(x))$**:** If $\mathsf{state}_{\mathsf{ORAM}} = (\mathsf{start}, *)$, set $\mathsf{state}_{\mathsf{ORAM}} = (\mathsf{start}, x)$, submit query $\mathsf{Execute}(x)$ to oracle, and receive (z, t). Set $\mathsf{out} = z$ and $c = t$.
Forward (g, f) to $\mathcal{S}_{\mathsf{code}}$. Upon receiving $\mathcal{S}_{\mathsf{code}}$'s output, $(\ell, \mathcal{I}, \vec{w})$, forward ℓ to $\mathcal{A}$.

Case: $\mathcal{I} \neq [n]$**.** Execute a $\mathsf{doNext}(x)$ instruction w.r.t. $\langle \mathsf{oCompNext}(\Pi), \widetilde{D} \rangle$.
Let $I_1, \ldots I_{\widetilde{\ell}}$ be the sequence of instructions executed by $\mathsf{doNext}(x)$. Recall that the first $\widetilde{\ell}_1$ instructions are reads, the next $\widetilde{\ell}_2$ instructions are writes, $\widetilde{\ell}_1 + \widetilde{\ell}_2 + 1 = \widetilde{\ell}$ and that $\widetilde{\ell}$, $\widetilde{\ell}_1$, $\widetilde{\ell}_2$ are public.
Let $\vec{v} = v_1, \ldots, v_{\widetilde{\ell}-1}$ be the vector of read/write locations corresponding to $I_1, \ldots I_{\widetilde{\ell}}$.
For $1 \leq i \leq \widetilde{\ell}_1$, do the following:

- If $d_{\mathsf{ORAM}} = \perp$ then abort.
- Output $S^{\text{DEC}}_{v_i}$ to $\mathcal{A}$, where $S^{\text{DEC}}_{v_i}$ be the ordered set of memory access locations corresponding to $\text{DEC}(v_i)$. If $v_i \in \mathcal{I}$, set $d_{\mathsf{ORAM}} = \perp$.

For $\widetilde{\ell}_1 + 1 \leq i \leq \widetilde{\ell}_1 + \widetilde{\ell}_2$, $\mathcal{S}$ does the following:

- If $d_{\mathsf{ORAM}} = \perp$ then abort.
- Output $S^{\text{UPDATE}}_{v_i}$ to $\mathcal{A}$, where $S^{\text{UPDATE}}_{v_i}$ be the ordered set of memory access locations corresponding to $\text{UPDATE}(v_i)$. Play the part of the updater interacting with $\mathcal{S}_{\mathsf{code}}$ and submit index v to $\mathcal{S}_{\mathsf{code}}$.

Set $c := c - 1 - \sigma \cdot (\widetilde{\ell}_1 + \widetilde{\ell}_2)$, where σ is the number of instructions in a DEC, UPDATE. If $c = 0$, output out to $\mathcal{A}$ and set $\mathsf{state}_{\mathsf{ORAM}} = (\mathsf{start}, *)$.
Case: $\mathcal{I} = [n]$**.** Do the following until termination:

1. If $d_{\mathsf{ORAM}} = \perp$ then abort.
2. Compute $(I, \mathsf{state}'_{\mathsf{ORAM}}) \leftarrow \mathsf{oCompNext}(\Pi)(\mathsf{state}_{\mathsf{ORAM}}, d_{\mathsf{ORAM}})$. Set $\mathsf{state}_{\mathsf{ORAM}} := \mathsf{state}'_{\mathsf{ORAM}}$.
3. If $I = (\mathsf{wait})$ then set $\mathsf{state}_{\mathsf{ORAM}} := 0^{\rho}$, $d_{\mathsf{ORAM}} := 0^{r}$ and terminate.
4. If $I = (\mathsf{stop}, z)$ then set $\mathsf{state}_{\mathsf{ORAM}} = (\mathsf{start}, *)$, $d := 0^{r}$, output z to $\mathcal{A}$ and terminate.
5. If $I = (\mathsf{read}, v, \perp)$ then set $d_{\mathsf{ORAM}} = \vec{w}_v$. Output S^{DEC}_{v} to $\mathcal{A}$.
6. If $I = (\mathsf{write}, v, d')$ then do the following: output S^{UPDATE}_{v} to $\mathcal{A}$. Play the part of the updater interacting with $\mathcal{S}_{\mathsf{code}}$ and submit index v to $\mathcal{S}_{\mathsf{code}}$.

Lemma 4.6. *Assume* $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$ *and* $\mathsf{NMCode} = (\text{ENC}, \text{DEC}, \text{UPDATE})$ *are as in Theorem 4.5. Let* Π *be the universal RAM next instruction function. For any* PPT *adversary* $\mathcal{A}$*, and any initial database* $D \in \{0,1\}^{\text{poly}(k)}$ *we have*

$$\mathbf{TamperExec}(\mathcal{A}, \mathcal{F}, \mathcal{G}, \langle \mathsf{CompNext}(\Pi), \mathsf{CompMem}(D)\rangle) \approx \mathbf{IdealExec}(\mathcal{S}, \langle \Pi, D\rangle)$$

To prove Lemma 4.6 we consider the sequence of hybrids H_0, H_1, $H_{1.5}$, H_2, defined below. We denote by $\mathsf{out}^k_{\mathcal{A},H_i}$, the output distribution of the adversary $\mathcal{A}$ on input security parameter k in Hybrid H_i, for $i \in \{0, 1, 1.5, 2\}$.

Hybrid H_0**:** This is the simulated experiment $\mathbf{IdealExec}(\mathcal{S}, \langle \Pi, D\rangle)$.
Hybrid H_1**:** This hybrid is the same as Hybrid H_0 except for the following change is made to the simulator's algorithm: in the setup stage, the real database D is used to compute $\widetilde{D} \leftarrow \mathsf{oCompMem}(D)$ (instead of $\widetilde{D} \leftarrow \mathsf{oCompMem}(D_0)$).

Claim 4.7.

$$\{\mathsf{out}^k_{\mathcal{A},H_0}\}_{k\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{out}^k_{\mathcal{A},H_1}\}_{k\in\mathbb{N}}.$$

This follows from the security of the ORAM scheme $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$. Details follow.

Proof. The only difference between the two Hybrids is that in Hybrid H_0 when $\mathsf{doNext}(x)$ is executed, the vector $\vec{v} = v_1, \ldots, v_{\widetilde{\ell}-1}$ is computed using the result of a $\mathsf{doNext}(x)$

instruction w.r.t. $\langle \mathsf{oCompNext}(\Pi), \widetilde{D}\rangle$, where $\widetilde{D} \leftarrow \mathsf{oCompMem}(D_0)$ (and D_0 is the dummy database). On the other hand, in Hybrid H_1, the vector $\vec{v} = v_1, \ldots, v_{\widetilde{\ell}-1}$ is computed using the result of a $\mathsf{doNext}(x)$ instruction w.r.t. $\langle \mathsf{oCompNext}(\Pi), \widetilde{D}\rangle$, where $\widetilde{D} \leftarrow \mathsf{oCompMem}(D)$ (and D is the real initial database). Thus, a distinguisher for Hybrids H_0 and H_1 immediately yields a distinguisher breaking the access-pattern hiding property of $\mathsf{ORAM} = (\mathsf{oCompMem}, \mathsf{oCompNext})$. □

Hybrid $H_{1.5}$**:** We consider the following modification of the Hybrid H_1 experiment:

Upon a $\mathsf{doNext}(x)$ query submitted by the adversary $\mathcal{A}$. If $\mathcal{I} \neq [n]$, execute the following code: (otherwise, the experiment remains unchanged):
Do the following until termination:

1. If $d_{\mathsf{ORAM}} = \perp$ then abort.
2. Compute $(I, \mathsf{state}'_{\mathsf{ORAM}}) \leftarrow \mathsf{oCompNext}(\Pi)(\mathsf{state}_{\mathsf{ORAM}}, d_{\mathsf{ORAM}})$. Set $\mathsf{state}_{\mathsf{ORAM}} := \mathsf{state}'_{\mathsf{ORAM}}$.
3. If $I = (\mathsf{wait})$ then set $\mathsf{state}_{\mathsf{ORAM}} := 0^{\rho}$, $d_{\mathsf{ORAM}} := 0^{r}$ and terminate.
4. If $I = (\mathsf{stop}, z)$ then set $\mathsf{state}_{\mathsf{ORAM}} := (\mathsf{start}, *)$, $d := 0^{r}$ and terminate with output z.
5. If $I = (\mathsf{read}, v, \perp)$ then if $v \notin \mathcal{I}$, set $d_{\mathsf{ORAM}} = \widetilde{D}[v]$. Otherwise, set $d_{\mathsf{ORAM}} = \perp$. Let S_v^{DEC} be the ordered set of memory access locations corresponding to $\mathrm{DEC}(v)$. Output S_v^{DEC} to $\mathcal{A}$.
6. If $I = (\mathsf{write}, v, d')$ then do the following: $\mathcal{S}$ plays the part of the updater interacting with $\mathcal{S}_{\mathsf{code}}$ and submits index v to $\mathcal{S}_{\mathsf{code}}$. Let S_v^{UPDATE} be the ordered set of memory access locations corresponding to $\mathrm{UPDATE}(v)$. Output S_v^{UPDATE} to $\mathcal{A}$.

Claim 4.8.

$$\{\mathsf{out}^k_{\mathcal{A},H_1}\}_{k\in\mathbb{N}} \equiv \{\mathsf{out}^k_{\mathcal{A},H_{1.5}}\}_{k\in\mathbb{N}}.$$

Proof. Intuitively, the difference between Hybrid H_1 and $H_{1.5}$ is that in H_1 in each doNext query, the memory locations $\vec{v} = v_1, \ldots, v_{\widetilde{\ell}-1}$ are pre-computed, whereas in $H_{1.5}$, the memory locations $v_1, \ldots, v_{\widetilde{\ell}-1}$ are computed on the fly. In particular, in H_1, the addresses $\vec{v}$ are computed assuming that each instruction of the form $(\mathsf{read}, v_i, \perp)$ sets d_{ORAM} to the correct value $d_{\mathsf{ORAM}} = \widetilde{D}[v_i]$. On the other hand, in $H_{1.5}$, d_{ORAM} may *not* be set to $\widetilde{D}[v_i]$. However, since we are in the case where $\mathcal{I} \neq [n]$, the only way this can happen is if $v_i \in \mathcal{I}$, in which case d_{ORAM} is set to $\perp$. But now, if $v_i \in \mathcal{I}$, then d_{ORAM} is set to $\perp$ in *both* H_1 and $H_{1.5}$ when the corresponding instruction $(\mathsf{read}, v_i, \perp)$ is simulated. Moreover, once d_{ORAM} is set to $\perp$ then the execution immediately aborts in both H_1 and $H_{1.5}$. Thus, the view of the adversary is identical in H_1 and $H_{1.5}$. □

Hybrid H_2 **:** This is the real experiment $\mathbf{TamperExec}(\mathcal{A}, F, G, \langle \mathsf{CompNext}(\Pi), \mathsf{CompMem}(D)\rangle)$.

Claim 4.9.

$$\{\mathsf{out}^k_{\mathcal{A},H_1}\}_{k\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{out}^k_{\mathcal{A},H_2}\}_{k\in\mathbb{N}}.$$

This follows from the security of the locally decodable and updatable code $\mathsf{NMCode} = (\mathrm{ENC}, \mathrm{DEC}, \mathrm{UPDATE})$. Details follow.

Proof. We claim that Hybrid $H_{1.5}$ can be perfectly simulated given the output of $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$, while Hybrid H_2 can be perfectly simulated given the output of $\mathbf{TamperLeak}_{\mathcal{A}',\mathcal{U},M}$, where $\mathcal{A}' = \mathcal{A}$ and $\mathcal{S} = \mathcal{S}$ and $\mathcal{U}$ is the following updater:

The Updater $\mathcal{U}$:

- $\mathcal{U}$ keeps persistent state $\mathsf{state}_{\mathsf{ORAM}}$ which is initialized to $(\mathsf{start}, *)$ and d_{ORAM} which is initialized to 0^r.
- On input $\widetilde{D}$, $\mathcal{U}$ does the following:
- If $d_{\mathsf{ORAM}} = \perp$, then $\mathcal{U}$ aborts.
- Otherwise, $\mathcal{U}$ computes $(I, \mathsf{state}'_{\mathsf{ORAM}}) := \mathsf{oCompNext}(\Pi)(\mathsf{state}_{\mathsf{ORAM}}, d_{\mathsf{ORAM}})$ and sets $\mathsf{state}_{\mathsf{ORAM}} := \mathsf{state}'_{\mathsf{ORAM}}$.
- If I is of the form $(\mathsf{read}, v, \perp)$, then $\mathcal{U}$ sets $d_{\mathsf{ORAM}} = \widetilde{D}[v]$ and outputs $\perp$.
- If I is of the form (write, v, d), then $\mathcal{U}$ outputs (v, d).
- Otherwise, $\mathcal{U}$ outputs $\perp$.

Thus, indistinguishability of hybrids $H_{1.5}$ and H_2 reduces to indistinguishability of $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$ and $\mathbf{TamperLeak}_{\mathcal{A}',\mathcal{U},M}$. This concludes the proof of Claim 4.9. □

Acknowledgements

We thank Yevgeniy Dodis for helpful discussions.

References

[1] D. Aggarwal, S. Agrawal, D. Gupta, H.K. Maji, O. Pandey, M. Prabhakaran. Optimal computational split-state non-malleable codes, in E. Kushilevitz, T. Malkin, editors, *TCC 2016-A, Part II*. LNCS, vol. 9563 (Springer, Heidelberg, 2016), pp. 393–417

[2] D. Aggarwal, Y. Dodis, T. Kazana, M. Obremski. Non-malleable reductions and applications, in R.A. Servedio, R. Rubinfeld, editors, *47th ACM STOC* (ACM Press, 2015), pp. 459–468

[3] D. Aggarwal, Y. Dodis, S. Lovett. Non-malleable codes from additive combinatorics, in D.B. Shmoys, editor, *46th ACM STOC* (ACM Press, 2014), pp. 774–783

[4] D. Aggarwal, S. Dziembowski, T. Kazana, M. Obremski. Leakage-resilient non-malleable codes, in Y. Dodis, J.B. Nielsen, editors, *TCC 2015, Part I*. LNCS, vol. 9014 (Springer, Heidelberg, 2015), pp. 398–426

[5] D. Aggarwal, T. Kazana, M. Obremski. Inception makes non-malleable codes stronger. *IACR Cryptol. ePrint Arch.* **2015**, 1013 (2015)

[6] D. Agrawal, B. Archambeault, J.R. Rao, P. Rohatgi. The EM side-channel(s), in B.S. Kaliski Jr., Ç Kaya Koç, C. Paar, editors, *CHES 2002*. LNCS, vol. 2523 (Springer, Heidelberg, 2003), pp. 29–45

[7] S. Agrawal, D. Gupta, H.K. Maji, O. Pandey, M. Prabhakaran. Explicit non-malleable codes against bit-wise tampering and permutations, in R. Gennaro, and M.J.B. Robshaw, editors, *CRYPTO 2015, Part I*. LNCS, vol. 9215 (Springer, Heidelberg, 2015), pp. 538–557

[8] S. Agrawal, D. Gupta, H.K. Maji, O. Pandey, M. Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations, in Y. Dodis, J.B. Nielsen, editors, *TCC 2015, Part I*. LNCS, vol. 9014 (Springer, Heidelberg, 2015), pp. 375–397

[9] M. Bellare, C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm, in T. Okamoto, editor, *ASIACRYPT 2000*. LNCS, vol. 1976. (Springer, Heidelberg, 2000), pp. 531–545

[10] M. Bellare, P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography, in T. Okamoto, editor, *ASIACRYPT 2000*. LNCS, vol. 1976 (Springer, Heidelberg, 2000), pp. 317–330

[11] E. Biham, A. Shamir. Differential fault analysis of secret key cryptosystems, in B.S. Kaliski Jr., editor, *CRYPTO'97*. LNCS, vol. 1294 (Springer, Heidelberg, 1997), pp. 513–525

[12] D. Boneh, R.A. DeMillo, R.J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptol.* **14**(2), 101–119 (2001)

[13] N. Chandran, B. Kanukurthi, R. Ostrovsky. Locally updatable and locally decodable codes, in Y. Lindell, editor, *TCC 2014*. LNCS, vol. 8349 (Springer, Heidelberg, 2014), pp. 489–514

[14] N. Chandran, B. Kanukurthi, S. Raghuraman. Information-theoretic local non-malleable codes and their applications, in E. Kushilevitz, T. Malkin, editors, *TCC 2016-A, Part II*. LNCS, vol. 9563 (Springer, Heidelberg, 2016), pp. 367–392

[15] M. Cheraghchi, V. Guruswami. Capacity of non-malleable codes, in M. Naor, editor, *ITCS 2014* (ACM, 2014), pp. 155–168

[16] M. Cheraghchi, V. Guruswami. Non-malleable coding against bit-wise and split-state tampering, in Y. Lindell, editor, *TCC 2014*. LNCS, vol. 8349 (Springer, Heidelberg, 2014), pp. 440–464

[17] S.G. Choi, A. Kiayias, T. Malkin. BiTR: built-in tamper resilience, in D.H. Lee, X. Wang, editors, *ASIACRYPT 2011*. LNCS, vol. 7073 (Springer, Heidelberg, 2011), pp. 740–758

[18] B. Chor, E. Kushilevitz, O. Goldreich, M. Sudan. Private information retrieval. *J. ACM* **45**(6), 965–981 (1998)

[19] S. Coretti, U. Maurer, B. Tackmann, D. Venturi. From single-bit to multi-bit public-key encryption via non-malleable codes, in Y. Dodis, J.B. Nielsen, editors, *TCC 2015, Part I*. LNCS, vol. 9014 (Springer, Heidelberg, 2015), pp. 532–560

[20] D. Dachman-Soled, Y.T. Kalai. Securing circuits against constant-rate tampering, in R. Safavi-Naini, R. Canetti, editors, *CRYPTO 2012*. LNCS, vol. 7417 (Springer, Heidelberg, 2012), pp. 533–551

[21] D. Dachman-Soled, Y.T. Kalai. Securing circuits and protocols against 1/poly(k) tampering rate, in Y. Lindell, editor, *TCC 2014*. LNCS, vol. 8349 (Springer, Heidelberg, 2014), pp. 540–565

[22] I. Damgård, S. Faust, P. Mukherjee, D. Venturi. Bounded tamper resilience: how to go beyond the algebraic barrier, in K. Sako, P. Sarkar, editors, *ASIACRYPT 2013, Part II*. LNCS, vol. 8270 (Springer, Heidelberg, 2013), pp. 140–160

[23] Y. Dodis, K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks, in T. Rabin, editor, *CRYPTO 2010*. LNCS, vol. 6223 (Springer, Heidelberg, 2010), pp. 21–40

[24] A. Duc, S. Dziembowski, S. Faust. Unifying leakage models: from probing attacks to noisy leakage, in P.Q. Nguyen, E. Oswald, editors, *EUROCRYPT 2014*. LNCS, vol. 8441 (Springer, Heidelberg, 2014), pp. 423–440

[25] S. Dziembowski, S. Faust. Leakage-resilient cryptography from the inner-product extractor, in D.H. Lee, X. Wang, editors, *ASIACRYPT 2011*. LNCS, vol. 7073 (Springer, Heidelberg, 2011), pp. 702–721

[26] S. Dziembowski, S. Faust. Leakage-resilient circuits without computational assumptions, in R. Cramer, editor, *TCC 2012*. LNCS, vol. 7194 (Springer, Heidelberg, 2012), pp. 230–247

[27] S. Dziembowski, T. Kazana, M. Obremski. Non-malleable codes from two-source extractors, in R. Canetti, J.A. Garay, editors, *CRYPTO 2013, Part II*. LNCS, vol. 8043 (Springer, Heidelberg, 2013), pp. 239–257

[28] S. Dziembowski, K. Pietrzak. Leakage-resilient cryptography, in *49th FOCS* (IEEE Computer Society Press, 2008), pp. 293–302

[29] S. Dziembowski, K. Pietrzak, D. Wichs. Non-malleable codes, in A. Chi-Chih Yao, editor, *ICS 2010* (Tsinghua University Press, 2010), pp. 434–452

[30] S. Faust, P. Mukherjee, J.B. Nielsen, D. Venturi. Continuous non-malleable codes, in Y. Lindell, editor, *TCC 2014*. LNCS, vol. 8349 (Springer, Heidelberg, 2014), pp. 465–488

[31] S. Faust, P. Mukherjee, J.B. Nielsen, D. Venturi. A tamper and leakage resilient von neumann architecture, in J. Katz, editor, *PKC 2015*. LNCS, vol. 9020 (Springer, Heidelberg, 2015), pp. 579–603

[32] S. Faust, P. Mukherjee, D. Venturi, D. Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits, in P.Q. Nguyen, E. Oswald, editors, *EUROCRYPT 2014*. LNCS, vol. 8441 (Springer, Heidelberg, 2014), pp. 111–128

[33] S. Faust, K. Pietrzak, D. Venturi. Tamper-proof circuits: how to trade leakage for tamper-resilience, in L. Aceto, M. Henzinger, J. Sgall, editors, *ICALP 2011, Part I*. LNCS, vol. 6755 (Springer, Heidelberg, 2011), pp. 391–402

[34] S. Faust, T. Rabin, L. Reyzin, E. Tromer, V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases, in H. Gilbert, editor, *EUROCRYPT 2010*. LNCS, vol. 6110 (Springer, Heidelberg, 2010), pp. 135–156

[35] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, T. Rabin. Algorithmic tamper-proof (ATP) security: theoretical foundations for security against hardware tampering, in M. Naor, editor, *TCC 2004*. LNCS, vol. 2951 (Springer, Heidelberg, 2004), pp. 258–277

[36] O. Goldreich, R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM* **43**(3), 431–473 (1996)

[37] S. Goldwasser, G.N. Rothblum. Securing computation against continuous leakage, in T. Rabin, editor, *CRYPTO 2010*. LNCS, vol. 6223 (Springer, Heidelberg, 2010), pp. 59–79

[38] S. Goldwasser, G.N. Rothblum. How to compute in the presence of leakage, in *53rd FOCS* (IEEE Computer Society Press, 2012), pp. 31–40

[39] J.A. Halderman, S.D. Schoen, N. Heninger, W. Clarkson, W. Paul, J.A. Calandrino, A.J. Feldman, J. Appelbaum, E.W. Felten. Lest we remember: cold boot attacks on encryption keys, in *USENIX Security Symposium* (2008), pp. 45–60

[40] Y. Ishai, E. Kushilevitz. On the hardness of information-theoretic multiparty computation, in C. Cachin, J. Camenisch, editors, *EUROCRYPT 2004*. LNCS, vol. 3027 (Springer, Heidelberg, 2004), pp. 439–455

[41] Y. Ishai, M. Prabhakaran, A. Sahai, D. Wagner. Private circuits II: keeping secrets in tamperable circuits, in S. Vaudenay, editor, *EUROCRYPT 2006*. LNCS, vol. 4004 (Springer, Heidelberg, 2006), pp. 308–327

[42] Y. Ishai, A. Sahai, D. Wagner. Private circuits: securing hardware against probing attacks, in D. Boneh, editor, *CRYPTO 2003*. LNCS, vol. 2729 (Springer, Heidelberg, 2003), pp. 463–481

[43] A. Juma, Y. Vahlis. Protecting cryptographic keys against continual leakage, in T. Rabin, editor, *CRYPTO 2010*. LNCS, vol. 6223 (Springer, Heidelberg, 2010), pp. 41–58

[44] J. Katz, L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes, in *32nd ACM STOC* (ACM Press, 2000), pp. 80–86

[45] J. Katz, M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation, in B. Schneier, editor, *FSE 2000*. LNCS, vol. 1978 (Springer, Heidelberg, 2001), pp. 284–299

[46] A. Kiayias, Y. Tselekounis. Tamper resilient circuits: the adversary at the gates, in K. Sako, P. Sarkar, editors, *ASIACRYPT 2013, Part II*. LNCS, vol. 8270 (Springer, Heidelberg, 2013), pp. 161–180

[47] P.C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, in N. Koblitz, editor, *CRYPTO'96*. LNCS, vol. 1109 (Springer, Heidelberg, 1996), pp. 104–113

[48] P.C. Kocher, J. Jaffe, B. Jun. Differential power analysis, in M.J. Wiener, editor, *CRYPTO'99*. LNCS, vol. 1666 (Springer, Heidelberg, 1999), pp. 388–397

[49] D. Lie, C.A. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J.C. Mitchell, M. Horowitz. Architectural support for copy and tamper resistant software, in *ASPLOS* (2000), pp. 168–177

[50] F.-H. Liu, A. Lysyanskaya. Tamper and leakage resilience in the split-state model, in R. Safavi-Naini, R. Canetti, editors, *CRYPTO 2012*. LNCS, vol. 7417 (Springer, Heidelberg, 2012), pp. 517–532

[51] S. Micali, L. Reyzin. Physically observable cryptography (extended abstract), in M. Naor, editor, *TCC 2004*. LNCS, vol. 2951 (Springer, Heidelberg, 2004), pp. 278–296

[52] K. Pietrzak. A leakage-resilient mode of operation, in A. Joux, editor, *EUROCRYPT 2009*. LNCS, vol. 5479 (Springer, Heidelberg, 2009), pp. 462–482

[53] T. Ristenpart, E. Tromer, H. Shacham, S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, in E. Al-Shaer, S. Jha, A.D. Keromytis, editors, *ACM CCS 09* (ACM Press, 2009), pp. 199–212

[54] G.N. Rothblum. How to compute under AC^0 leakage without secure hardware, in R. Safavi-Naini, R. Canetti, editors, *CRYPTO 2012*. LNCS, vol. 7417 (Springer, Heidelberg, 2012), pp. 552–569

[55] G.E. Suh, D.E. Clarke, B. Gassend, M. van Dijk, S. Devadas. AEGIS: architecture for tamper-evident and tamper-resistant processing, in *Proceedings of the 17th Annual International Conference on Supercomputing, ICS 2003* (2003), pp. 160–171

[56] A. Vasudevan, J.M. McCune, J. Newsome, A. Perrig, L. van Doorn. CARMA: a hardware tamper-resistant isolated execution environment on commodity x86 platforms, in H. Youl Youm, Y. Won, editors, *ASIACCS 12* (ACM Press, 2012), pp. 48–49

[57] S. Yekhanin. Locally decodable codes. *Found. Trends Theor. Comput. Sci.* **6**(3), 139–255 (2012)

J Cryptol
https://doi.org/10.1007/s00145-018-9286-z

Journal of
CRYPTOLOGY

Leakage Resilience from Program Obfuscation

Dana Dachman-Soled*
University of Maryland, College Park, USA
danadach@ece.umd.edu

S. Dov Gordon†
George Mason University, Fairfax, USA
crypto@dovgordon.com

Feng-Hao Liu‡
Florida Atlantic University, Boca Raton, USA
fenghao.liu@fau.edu

Adam O'Neill
Georgetown University, Washington, DC, USA
adam@cs.georgetown.edu

Hong-Sheng Zhou
Virginia Commonwealth University, Richmond, USA
hszhou@vcu.edu

Communicated by Tal Rabin.

Received 23 August 2016 / Revised 3 March 2018

Abstract. The literature on leakage-resilient cryptography contains various leakage models that provide different levels of security. In the bounded leakage model (Akavia et al.—TCC 2009), it is assumed that there is a fixed upper bound L on the number of bits the attacker may leak on the secret key in the entire lifetime of the scheme. Alternatively, in the continual leakage model (Brakerski et al.—FOCS 2010, Dodis et al.—FOCS 2010), the lifetime of a cryptographic scheme is divided into "time periods" between which the scheme's secret key is updated. Furthermore, in its attack the adversary is allowed to obtain some bounded amount of leakage on the current secret key during each time period. In the continual leakage model, a challenging problem has been to provide security against *leakage on key updates*, that is, leakage that is a function of not only the current secret key but also the randomness used to update it. We propose a modular

*This work was done in part while the author was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF Grant #CNS-1523467. This work was supported in part by NSF CAREER Award #CNS-1453045 and by a Ralph E. Powe Junior Faculty Enhancement Award.

†This work was done in part when the author was a research scientist at Applied Communication Sciences.

‡This work was done in part when the author was a postdoc at the University of Maryland. Partial effort of the work is supported by the NSF Award #CNS-1657040.

Published online: 25 April 2018

approach to overcome this problem based on program obfuscation. Namely, we present a compiler that transforms any public key encryption or signature scheme that achieves a slight strengthening of continual leakage resilience, which we call *consecutive* continual leakage resilience, to one that is continual leakage resilient with leakage on key updates, assuming *indistinguishability obfuscation* (Barak et al.—CRYPTO 2001, Garg et al.—FOCS 2013). Under stronger forms of obfuscation, the leakage rate tolerated by our compiled scheme is essentially as good as that of the starting scheme. Our compiler is derived by making a connection between the problems of leakage on key updates and so-called sender-deniable encryption (Canetti et al.—CRYPTO 1997), which was recently constructed based on indistinguishability obfuscation by Sahai and Waters (STOC 2014). In the bounded leakage model, we give an approach to constructing leakage-resilient public key encryption from program obfuscation based on the public key encryption scheme of Sahai and Waters (STOC 2014). In particular, we achieve leakage-resilient public key encryption tolerating L bits of leakage for any L from iO and one-way functions. We build on this to achieve leakage-resilient public key encryption with optimal leakage rate of $1 - o(1)$ based on stronger forms of obfuscation and collision-resistant hash functions. Such a leakage rate is not known to be achievable in a generic way based on public key encryption alone. We then develop additional techniques to construct public key encryption that is (consecutive) continual leakage resilient under appropriate assumptions, which we believe is of independent interest.

Keywords. Indistinguishability obfuscation, Leakage resilience, Public key encryption, Digital signatures.

1. Introduction

1.1. *Background and Motivation*

In recent years, researchers have uncovered a variety of ways to capture cryptographic keys through *side-channel attacks*: physical measurements, such as execution time, power consumption, and even sound waves generated by the processor. This has prompted cryptographers to build models for these attacks and to construct *leakage-resilient* schemes that remain secure in the face of such attacks. Of course, if the adversary can leak the entire secret key, security becomes impossible, and so the "bounded" leakage model was introduced (cf. [2,11,39,46]). Here, it is assumed that there is a fixed upper bound, L on the number of bits the attacker may leak, regardless of the parameters of the scheme, or, alternatively, it is assumed that the attacker is allowed to leak $L = \lambda \cdot |\mathsf{sk}|$ total number of bits, where the amount of leakage increases as the size of the secret key increases. Various works constructed public key encryption and signature schemes with optimal leakage rate of $\lambda = 1 - o(1)$, from specific assumptions (cf. [11,46]). Hazay et al. [35] even constructed a leakage-resilient public key encryption scheme in this model assuming only the existence of standard public key encryption, although the leakage rate achieved by their scheme was not optimal.[1]

[1] In the construction of Hazay et al. [35], the secret key of the constructed scheme consists of n secret keys of the underlying public key encryption scheme, where each underlying secret key is randomly selected from a set of size m and m is polynomial in security parameter. The total amount of leakage that can be tolerated is approximately $n \log(m)$ bits. Thus, the leakage rate is $\frac{n \log(m)}{n \cdot s} \in \Theta(\frac{\log(m)}{s})$, where s denotes the length of the secret key of the underlying scheme.

Surprisingly, it is possible to do better; a strengthening of the model—the "continual" leakage model[2]—allows the adversary to request *unbounded* leakage. This model was introduced by Brakerski et al. [12]—who constructed continual leakage-resilient (CLR) public key encryption and signature schemes—and Dodis et al. [21]—who constructed CLR signature schemes. Intuitively, the CLR model divides the lifetime of the attack, which may be unbounded, into time periods and: (1) allows the adversary to obtain the output of a "bounded" leakage function in each time period and (2) allows the secret key (but not the public key!) to be updated between time periods. So, while the adversary's leakage in each round is bounded, the total leakage is unbounded.

Note that the algorithm used by any CLR scheme to update the current secret key to the next one must be *randomized*, since otherwise the adversary can obtain some future secret key, bit by bit, via its leakage in each time period. While the CLR schemes of [12,21] were able to tolerate a $1 - o(1)$ leakage rate, handling leakage *during the update procedure itself*—that is, produced as a function of the randomness used by the update algorithm as well as the current secret key—proved to be much more challenging. The first substantial progress on this problem of "leakage on key updates" was made by Lewko et al. [43], with their techniques being considerably refined and generalized by Dodis et al. [24]. In particular, they give encryption and signature schemes that are CLR with leakage on key updates tolerating a constant leakage rate, using "dual-system" techniques (cf. [48]) in bilinear groups.

1.2. *Overview of Our Results*

Our first main contribution is to show how to compile any public key encryption or signature scheme that satisfies a slight strengthening of CLR (which we call "consecutive" CLR or 2CLR) *without* leakage on key updates to one that is CLR *with* leakage on key updates. Our compiler is based on a new connection we make between the problems of leakage on key updates and "sender deniability" [13] for encryption schemes. In particular, our compiler uses program obfuscation—either indistinguishability obfuscation (iO) [5,29] or the public-coin differing-inputs obfuscation (diO) [37][3]—and adapts and extends techniques recently developed by Sahai and Waters [47] to achieve sender-deniable encryption. This demonstrates the applicability of the techniques of [47] to other seemingly unrelated contexts.[4] We then show that the existing CLR encryption scheme of Brakerski et al. [12] can be extended to meet the stronger notion of 2CLR that we require for our compiler. Additionally, we show all our results carry over to signatures as well. In particular, we show that 2CLR PKE implies 2CLR signatures (via

[2] Here "continual" refers to the fact that the total amount of leakage obtained by the adversary is unbounded. Additionally, the model is more accurately called the continual *memory* leakage model to contrast with schemes constructed under an assumption that "only computation leaks" [45].

[3] To the best of our knowledge, no impossibility results are known for public-coin differing-inputs obfuscation. Indeed, the impossibility results of Garg et al. [30] do not apply to this setting. In either case, current constructions rely on multilinear maps, whose first candidate construction was given by [28].

[4] We note that the techniques of [47] have been shown useful in adaptively secure two-party and multiparty computation [14,18,31] and "only computation leaks" (OCL) circuits without trusted hardware [19]. We note that this work precedes the work of [18].

the intermediate notion of CLR "one-way relations" of Dodis et al. [21]), and observe that our compiler also upgrades 2CLR signatures to ones that are CLR with leakage on updates.

Our second main contribution concerns constructions of leakage-resilient public key encryption directly from obfuscation. In particular, we show that the approach of Sahai and Waters to achieve public key encryption from iO and punctured pseudorandom functions [47] can be extended to achieve leakage resilience in the bounded leakage model. Specifically, we achieve (1) leakage-resilient public key encryption tolerating L bits of leakage for any L from iO and one-way functions, (2) leakage-resilient public key encryption with optimal leakage rate of $1 - o(1)$ based on public-coin differing-inputs obfuscation and collision-resistant hash functions, and (3) (consecutive) CLR public key encryption with constant (although not optimal, on the order of one over several hundred) leakage rate from differing-inputs obfuscation (not public coin) and standard assumptions. Extending the construction from (2) to achieve continual leakage resilience, without these additional assumptions, is an interesting open problem.

1.3. *Summary and Perspective*

In summary, we provide a thorough study of the connection between program obfuscation and leakage resilience. We define a new notion of leakage resilience (2CLR) and demonstrate new constructions of 2CLR-secure encryption and signature schemes from program obfuscation. Also using program obfuscation, we construct a compiler that lifts 2CLR-secure schemes to CLR with leakage on key updates; together with our new constructions, this provides a unified and modular method for constructing CLR with leakage on key updates. Under appropriate assumptions (namely the ones used by Brakerski et al. [12] in their construction), this approach allows us to achieve a leakage rate of $1/4 - o(1)$ with leakage on key updates, a large improvement over prior work, where the best leakage rate was $1/258 - o(1)$ [43]. Our result nearly matches the trivial upper bound of $1/2 - o(1)$.[5] In the bounded leakage model, we show that it is possible to achieve optimal-rate leakage-resilient public key encryption from obfuscation and generic assumptions.

Comparing our results in the bounded leakage model with the work of Hazay et al. [35], we have (1) leakage-resilient public key encryption tolerating L bits of leakage from iO and one-way functions and (2) leakage-resilient public key encryption with optimal leakage rate based on public-coin differing-inputs obfuscation and collision-resistant hash functions. As we mentioned above, Hazay et al. [35] constructed bounded leakage-resilient public key encryption in the bounded leakage model from a far weaker generic assumption (they require only standard public key encryption). Moreover, the leakage rate of Hazay et al. [35] is far better than the leakage rate we achieve in (1), since in our iO-based construction, the secret key consists of an entire obfuscated program,

[5] Unlike the case of CLR without leakage on key updates, observe that any scheme that is CLR with leakage on key updates can leak at most $1/2 \cdot |\mathsf{sk}|$-bits per time period, since otherwise the adversary can recover an entire secret key. As a consequence, the optimal leakage rate for a scheme that is CLR with leakage on key updates is at most $\frac{1/2 \cdot |\mathsf{sk}|}{|\mathsf{sk}| + |r_{up}|} < 1/2$, where $|\mathsf{sk}|$ is the secret key length and $|r_{up}|$ is the length of the randomness needed by the update algorithm.

which will be extremely large. Thus, the work of Hazay et al. [35] completely subsumes (1). On the other hand, the leakage rate we achieve in (2) is optimal and so in this case, our leakage rate improves upon the rate of Hazay et al. [35], though we require the far stronger assumption of public-coin differing-inputs obfuscation for our result.

Finally, we discuss our result in the continuous leakage model on (3) (consecutive) CLR public key encryption with constant leakage rate from differing-inputs obfuscation and standard assumptions. When instantiating our construction in (3), the assumptions and parameters achieved are inferior to those of the Brakerski et al. [12] scheme (which we adapt to our setting). Our intention in (3) is therefore to explore what can be done from generic assumptions, ideally showing that (consecutive) CLR public key encryption can be constructed from any PKE scheme and diO. Unfortunately, we fall somewhat short, requiring that the underlying encryption scheme posses various additional properties.

Given the above discussion, we feel that the main value of our results in the bounded leakage model is that they provide direct insight into the connection between obfuscation and leakage resilience. We are also hopeful that our techniques in the continual model might lead to future improvements in rate as well as a better understanding of the relationship between obfuscation and continual leakage resilience.

1.4. *Details and Techniques*

Part I: The Leak-on-Update Compiler. As described above, in the model of continual leakage resilience (CLR) [12,21] for public key encryption or signature schemes, the secret key can be updated periodically (according to some algorithm Update) and the adversary can obtain bounded leakage between any two updates. Our compiler applies to schemes that satisfy a slight strengthening of CLR we call *consecutive* CLR, where the adversary can obtain bounded leakage as a *joint* function of any two consecutive keys. More formally, let $\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_t, \ldots$ be the secret keys at each time period, where $\mathsf{sk}_i = \mathsf{Update}(\mathsf{sk}_{i-1}, r_i)$, and each r_i denotes fresh random coins used at that round. For leakage functions $f_1, \ldots, f_t, \ldots$ (chosen adaptively by the adversary), consider the following two leakage models:

(1) For consecutive CLR (2CLR), the adversary obtains leakage

$$f_1(\mathsf{sk}_0, \mathsf{sk}_1), f_2(\mathsf{sk}_1, \mathsf{sk}_2), \ldots, f_t(\mathsf{sk}_{t-1}, \mathsf{sk}_t), \ldots .$$

(2) For CLR with leakage on key updates, the adversary obtains leakage

$$f_1(\mathsf{sk}_0, r_1), f_2(\mathsf{sk}_1, r_2), \ldots, f_t(\mathsf{sk}_{t-1}, r_t), \ldots .$$

Our compiler from 2CLR to CLR with leakage on key updates produces a slightly different Update algorithm for the compiled scheme depending on whether we assume indistinguishability obfuscation (iO) [5,29] or public-coin differing-inputs obfuscation [37]. In both cases, if we start with an underlying scheme that is consecutive two-key CLR while allowing μ-bits of leakage, then our compiled scheme is CLR with leakage on key updates with leakage rate

$$\frac{\mu}{|\mathsf{sk}| + |r_{up}|} ,$$

where $|r_{up}|$ is the length of the randomness required by Update. When using iO, we obtain $|r_{up}| = 5|\mathsf{sk}|$, where $|\mathsf{sk}|$ is the secret key length for the underlying 2CLR scheme, whereas using public-coin differing-inputs obfuscation we obtain $|r_{up}| = |\mathsf{sk}|$. Thus:

- Assuming iO, the compiled scheme is CLR with leakage on key updates with leakage rate $\frac{\mu}{6 \cdot |\mathsf{sk}|}$.
- Assuming public-coin differing-inputs obfuscation, the compiled scheme is CLR with leakage on key updates with leakage rate $\frac{\mu}{2 \cdot |\mathsf{sk}|}$.

Thus, if the underlying 2CLR scheme tolerates the optimal number of bits of leakage ($\approx 1/2 \cdot |\mathsf{sk}|$), then our resulting public-coin differing-inputs-based scheme achieves leakage rate $1/4 - o(1)$.

Our compiler is obtained by adapting and extending the techniques developed by [47] to achieve sender-deniable PKE from any PKE scheme. In sender-deniable PKE, a sender, given a ciphertext and *any* message, is able to produce coins that make it appear that the ciphertext is an encryption of that message. Intuitively, the connection we make to leakage on key updates is that the simulator in the security proof faces a similar predicament to the coerced sender in the case of deniable encryption; it needs to come up with some randomness that "explains" a current secret key as the update of an old one. Our compiler makes any two such keys explainable in a way that is similar to how Sahai and Waters make any ciphertext and message explainable. Intuitively, this is done by "encoding" a secret key in the explained randomness in a special way that can be detected only by the (obfuscated) Update algorithm. Once detected, the Update algorithm outputs the encoded secret key, instead of running the normal procedure.

However, in our context, naïvely applying their techniques would result in the randomness required by our Update algorithm being very long, which, as described above, affects the leakage rate of our resulting CLR scheme with leakage on key updates in a crucial way (we would not even be able to get a constant leakage rate). We decrease the length of this randomness in two steps. First, we note that the sender-deniable encryption scheme of Sahai and Waters encrypts a message bit by bit and "explains" each message bit individually. This appears to be necessary in their context in order to allow the adversary to choose its challenge messages *adaptively* depending on the public key. For our setting, this is not the case, since the secret key is chosen honestly (not by the adversary), so "non-adaptive" security is in fact sufficient in our context and we can "explain" a secret key all at once. This gets us to $|r_{up}| = 5 \cdot |\mathsf{sk}|$ and thus $1/12 - o(1)$ leakage rate assuming the underlying 2CLR scheme can tolerate the optimal leakage. Second, we observe that by switching assumptions from iO to the public-coin differing-inputs obfuscation we can replace some instances of sk in the explained randomness with its value under a collision-resistant hash, which gets us to $|r_{up}| = \mathsf{sk}$ and thus $1/4 - o(1)$ leakage rate in this case.

A natural question is whether the upper bound of $1/2 - o(1)$ leakage rate for CLR with leakage on key updates, can be attained via our techniques (if at all). We leave this as an intriguing open question, but note that the *only* way to do so would be to further decrease $|r_{up}|$ so that $|r_{up}| < |\mathsf{sk}|$.

Part II: Constructions against Two-key Consecutive Continual Leakage. We revisit the existing CLR public key encryption scheme of [12] and show that a suitable mod-

ification of it achieves 2CLR[6] with optimal $1/4 - o(1)$ leakage rate,[7] under the same assumption used by [12] to achieve optimal leakage rate in the basic CLR setting (namely the symmetric external Diffie–Hellman (SXDH) assumption in bilinear groups; smaller leakage rates can be obtained under weaker assumptions). Our main technical tool here is a new generalization of the Crooked Leftover Hash Lemma [6,26] that generalizes the result of [12], which shows that "random subspaces are leakage resilient," showing that random subspaces are in fact resilient to "consecutive leakage." Our claim also leads to a simpler analysis of the scheme than appears in [12].

Finally, we also show (via techniques from learning theory) that 2CLR public key encryption generically implies 2CLR one-way relations. Via a transformation of Dodis et al. [21], this then yields 2CLR signatures with the same leakage rate as the starting encryption scheme. Therefore, all the above results translate to the signature setting as well. We also show a direct approach to constructing 2CLR one-way relations following [21] based on the SXDH assumption in bilinear groups, although we are not able to achieve as good of a leakage rate this way (only $1/8 - o(1)$).

Part III: Exploring the relationship between (bounded and continual) leakage resilience and obfuscation. Note that, interestingly, even the strong notion of virtual black-box (VBB) obfuscation does not immediately lead to constructions of leakage-resilient public key encryption. In particular, if we replace the secret key of a public key encryption scheme with a VBB obfuscation of the decryption algorithm, it is not clear that we gain anything: For example, the VBB obfuscation may output a circuit of size $|C|$, where only $\sqrt{|C|}$ number of the gates are "meaningful" and the remaining gates are simply "dummy" gates, in which case we cannot hope to get a leakage bound better than $L = \sqrt{|C|}$, and a leakage rate of $1/\sqrt{|C|}$. Nevertheless, we are able to show that the PKE scheme of Sahai and Waters (SW) [47], which is built from iO and "punctured pseudo-random functions (PRFs)," can naturally be made leakage resilient. To give some brief intuition, a ciphertext in our construction is of the form $(r, w, \mathsf{Ext}(\mathsf{PRF}(k; r), w) \oplus m)$, where Ext is a strong extractor, r and w are random values,[8] and the PRF key k is embedded in obfuscated programs that are used in both encryption and decryption. In the security proof, we "puncture" the key k at the challenge point, t^*, and hardcode the mapping $t^* \to y$, where $y = \mathsf{PRF}(k; t^*)$, in order to preserve the input/output behavior. As in SW, we switch the mapping to $t^* \to y^*$ for a random y^* via security of the puncturable PRF. But now observe we have that the min-entropy of y^* is high even after leakage, so the output of the extractor is close to uniform. To achieve optimal leakage rate, we further modify the scheme to separate $t^* \to y^*$ from the obfuscated program and store only an encryption of $t^* \to y^*$ in the secret key.

Note that the last change lends itself to achieving (consecutive) CLR, since the secret key can be refreshed by re-randomizing the encryption. However, the information theo-

[6] Note that [12] also constructs such a signature scheme, but, as discussed below, such a signature scheme can in fact be generically obtained, and therefore, for simplicity we do not consider their direct construction here.

[7] In the 2CLR model, the maximum amount of leakage is roughly $1/2 \cdot |\mathsf{sk}|$, so the optimal rate is roughly $\frac{1/2 \cdot |\mathsf{sk}|}{|\mathsf{sk}| + |\mathsf{sk}|} = 1/4$.

[8] Technically, we actually use pseudorandom value r, just as SW do. We omit this here to make the explanation a little more clear.

retic argument above about the entropy remaining in y^* no longer holds, since additional entropy is lost in every round, and, eventually, y^* might be recovered in full. To address this issue, we must prevent the attacker from directly leaking on y^* in each round. Instead of embedding an encryption of $t^* \rightarrow y^*$ in the secret key, we embed an encryption of a tuple $(s_i, \alpha_i, H(t^*)) \rightarrow y^*$ using a fresh s_i in each round i, subject to the constraint that $\alpha_i = \langle s_i, t^* \rangle$. In order to determine whether to output y^* on some input t, our obfuscated circuit decrypts and checks whether $H(t^*) = H(t) \wedge \langle s_i, t \rangle = \alpha_i$, where H is a collision-resistant hash function. We rely on the following facts to ensure that y^* remains indistinguishable from random given the adversary's view: a) the adversary must form his leakage queries before learning t^*, b) very little information about t^* is contained in the secret key, and c) due to the previous facts, and since the inner product is a good two-source extractor, $\langle s_i, t^* \rangle$ remains very close to uniform, even under the leakage. It follows that we can switch, even under leakage, to a random α^*, uncorrelated with s_i, t^*. Since it is now hard to find inputs satisfying $H(t^*) = H(t) \wedge \langle s_i, t \rangle = \alpha^*$, we can, using security of the diO, *ignore* this conditional statement and replace y^*, with a 0 string in the secret key, while still using y^* in the challenge ciphertext.

In the above discussion, we omitted some additional technical challenges due to lack of space. Most notably, we also require that the encryption scheme used for encrypting the tuple in the secret key satisfies a notion of "diO-compatible RCCA-secure re-randomizability," which we introduce (see Sect. A.2), and show that the "controlled-malleable" RCCA-secure PKE due to Chase et al. [17] based on the Decision-Linear assumption in bilinear groups schemes satisfies it, giving us a constant leakage rate for our (2)CLR scheme. For an in-depth technical overview and complete proof, see Sect. A.

1.5. *Related Work*

Leakage-Resilient Cryptography. We discuss various types of memory leakage attacks that have been studied in the literature. Memory attacks are a strong type of attack, where all secrets in memory are subject to leakage, whether or not they are actively being computed on. Memory leakage attacks are motivated by the cold-boot attack of Halderman et al. [34], who showed that for some time after power is shut down, partial data can be recovered from random access memory (DRAM and SRAM). Akavia et al. [2] introduced the model of bounded memory attacks, where arbitrary leakage on memory is allowed, as long as the output size of the leakage function is bounded. Additional models introduced by [16,27] and [23] allow unbounded-length noisy leakage, unbounded-length leakage under restricted leakage functions, or unbounded-length hard-to-invert leakage, respectively. The works of [12] and [21] introduced the notion of "continual memory leakage" for public key primitives where the secret key is updated while the public key remains the same. This model allows bounded memory leakage between key refreshes. Finally, [12,21,24,43] considered the model of continual memory leakage with leak on update, where leakage can occur while the secret key is being updated. In this work, we consider bounded memory attacks, continual memory leakage and continual memory leakage with leak on update.

There is a long line of constructions of leakage-resilient cryptographic primitives, including public key encryption that are leakage resilient (LR) against bounded memory attacks [2,46]; public key encryption that is continual leakage resilient (CLR) without leak on update [12]; public key encryption that is CLR with leak on update [43]; digital signature schemes that are leakage resilient (LR) against bounded memory attacks [39]; digital signature schemes that are LR against bounded memory attacks on both secret key and random coins for signing [11,39,44]; digital signature schemes that are CLR without leak on update [21]; digital signature schemes that are CLR with leak on update [43].

Obfuscation and Its Applications. Since the breakthrough result of Garg et al. [29], demonstrating the first candidate of indistinguishability obfuscation (iO) for all circuits, a myriad of uses for iO in cryptography have been found. Among these results, the puncturing methodology by Sahai and Waters [47] has been found very useful. Related notions such as differing-inputs obfuscation (diO) [4] have been studied [3,9,37]. Please refer to [49,50] for new constructions, applications, and limitations of obfuscation.

1.6. *Organization*

We present definitions and preliminaries in Sect. 2. In Sect. 3, we present our compiler from 2CLR public key encryption/signatures to CLR public key encryption/signatures with leakage on key update. In Sect. 4, we prove that the public key encryption scheme of Brakerski et al. [12] achieves 2CLR. In Sect. 5, we present constructions of leakage-resilient public key encryption (in the non-continual setting) from obfuscation and generic assumptions. In Sect. 6, we define 2CLR security for one-way relations and prove that the construction of Dodis et al. [21] achieves the 2CLR notion. In Sect. 7, we present a construction of 2CLR signatures from 2CLR one-way relations. Finally, in Appendix A, we address the question of constructing 2CLR public key encryption from obfuscation and generic assumptions.

2. Definitions and Preliminaries

Statistical Indistinguishability. The statistical distance between two random variables X, Y is defined by

$$\Delta(X, Y) = \frac{1}{2} \sum_{x} |\Pr[X = x] - \Pr[Y = x]|$$

We write $X \overset{s}{\approx} Y$ to denote that the statistical distance is negligible in the security parameter, and we say that X, Y are statistically indistinguishable.

2.1. *Security Definitions for Leakage-Resilient Public Key Encryption*

In this subsection, we present the definitions of various leakage-resilient public key encryption schemes. These definitions are from the literature. In Subsect. 2.2, we present

the definitions for leakage-resilient signature schemes. Jumping ahead, in Subsect. 3.1, we start to present our new definition for consecutive continual leakage resilience (2CLR).

We present definitions for obfuscation and puncturable PRFs in Subsects. 2.3 and 2.4.

2.1.1. *One-Time Leakage Model*

A public key encryption scheme PKE consists of three algorithms: $\mathsf{PKE.Gen}$, $\mathsf{PKE.Enc}$, and $\mathsf{PKE.Dec}$.

- $\mathsf{PKE.Gen}(1^\kappa) \rightarrow (\mathsf{pk}, \mathsf{sk})$. The key generation algorithm takes in the security parameter κ and outputs a public key pk and a secret key sk.
- $\mathsf{PKE.Enc}(\mathsf{pk}, m) \rightarrow c$. The encryption algorithm takes in a public key pk and a message m. It outputs a ciphertext c.
- $\mathsf{PKE.Dec}(\mathsf{sk}, c) \rightarrow m$. The decryption algorithm takes in a ciphertext c and a secret key sk. It outputs a message m.

Correctness. The PKE scheme satisfies correctness if $\mathsf{PKE.Dec}(\mathsf{sk}, c) = m$ with all but negligible probability whenever $(\mathsf{pk}, \mathsf{sk})$ is produced by $\mathsf{PKE.Gen}$ and c is produced by $\mathsf{PKE.Enc}(\mathsf{pk}, m)$.

Security. We define one-time leakage-resilient security for PKE schemes in terms of the following game between a challenger and an attacker. (This extends the usual notion of semantic security to our leakage setting.) We let κ denote the security parameter, and the parameter μ controls the amount of leakage allowed.

Setup Phase.	The game begins with a setup phase. The challenger calls $\mathsf{PKE.Gen}(1^\kappa)$ to create the initial secret key sk and public key pk. It gives pk to the attacker. No leakage is allowed in this phase.
Query Phase.	The attacker specifies an efficiently computable leakage function f, whose output is at most μ bits. The challenger returns $f(\mathsf{sk})$ to the attacker. We sometimes refer to the challenger as a stateful, "leakage oracle," denoted $\mathcal{O}$, during the query phase of the security experiment.
Challenge Phase.	The attacker chooses two messages m_0, m_1 which it gives to the challenger. The challenger chooses a random bit $b \in \{0, 1\}$, encrypts m_b, and gives the resulting ciphertext to the attacker. The attacker then outputs a guess b' for b. The attacker wins the game if $b = b'$. We define the advantage of the attacker in this game as $\lvert\frac{1}{2} - \Pr[b' = b]\rvert$.

Definition 1. (*One-time Leakage Resilience*) We say a public key encryption scheme is μ-leakage resilient against one-time key leakage if any probabilistic polynomial-time attacker only has a negligible advantage (negligible in κ) in the above game.

2.1.2. *Continual Leakage Model*

In the continual leakage setting, we require an additional algorithm $\mathsf{PKE.Update}$ which updates the secret key. Specifically, the update algorithm takes in a secret key sk_{i-1} and some randomness r_i, and produces a new secret key sk_i for the *same* public key.

Thus, scheme PKE consists of four algorithms: PKE.Gen, PKE.Enc, PKE.Dec, and PKE.Update.

- $\mathsf{PKE.Gen}(1^\kappa) \rightarrow (\mathsf{pk}, \mathsf{sk}_0)$. The key generation algorithm takes in the security parameter and outputs a public key pk and a secret key sk_0.
- $\mathsf{PKE.Enc}(\mathsf{pk}, m) \rightarrow c$. The encryption algorithm takes in a public key pk and a message m. It outputs a ciphertext c.
- $\mathsf{PKE.Dec}(\mathsf{sk}_i, c) \rightarrow m$. The decryption algorithm takes in a ciphertext c and a secret key sk_i. It outputs a message m.
- $\mathsf{PKE.Update}(\mathsf{sk}_{i-1}) \rightarrow \mathsf{sk}_i$. The update algorithm takes in a secret key sk_{i-1} and produces a new secret key sk_i for the *same* public key. Here some randomness r_i is used in the update algorithm.

Correctness. The PKE scheme satisfies correctness if $\mathsf{PKE.Dec}(\mathsf{sk}_i, c) = m$ with all but negligible probability whenever pk and sk are produced by PKE.Gen, sk_i is obtained by calls to PKE.Update on previously obtained secret keys (starting with sk_0), and c is produced by $\mathsf{PKE.Enc}(\mathsf{pk}, m)$.

Security. We define continual leakage-resilient security for PKE schemes in terms of the following game between a challenger and an attacker. (This extends the usual notion of semantic security to our leakage setting.) We let κ denote the security parameter, and the parameter μ controls the amount of leakage allowed.

Setup Phase. The game begins with a setup phase. The challenger calls $\mathsf{PKE.Gen}(1^\kappa)$ to create the initial secret key sk_0 and public key pk. It gives pk to the attacker. No leakage is allowed in this phase.

Query Phase. In this phase, the attacker launches a polynomial number of leakage queries. Each time, say in the ith query, the attacker provides an efficiently computable leakage function f_i whose output is at most μ bits, and the challenger chooses randomness r_i, updates the secret key from sk_{i-1} to sk_i, and gives the attacker the leakage response ℓ_i. In the *regular continual leakage* model, the leakage attack is applied on a single secret key, and the leakage response $\ell_i = f_i(\mathsf{sk}_{i-1})$. In the *continual leak-on-update* model, the leakage attack is applied on the current secret key and the randomness used for updating the secret key, i.e., $\ell_i = f_i(\mathsf{sk}_{i-1}, r_i)$. We sometimes refer to the challenger as a stateful, "leakage oracle," denoted $\mathcal{O}$, during the query phase of the security experiment.

Challenge Phase. The attacker chooses two messages m_0 and m_1 which it gives to the challenger. The challenger chooses a random bit $b \in \{0, 1\}$, encrypts m_b, and gives the resulting ciphertext to the attacker. The attacker then outputs a guess b' for b. The attacker wins the game if $b = b'$. We define the advantage of the attacker in this game as $|\frac{1}{2} - \Pr[b' = b]|$.

Definition 2. (*Continual Leakage Resilience*) We say a public key encryption scheme is μ-CLR secure (respectively, μ-CLR secure with leakage on key updates) if any PPT attacker only has a negligible advantage (negligible in κ) in the above game.

2.2. *Leakage-Resilient Signatures*

A digital signature scheme SIG consists of three algorithms: $\mathsf{SIG.Gen}$, $\mathsf{SIG.Sign}$, and $\mathsf{SIG.Verify}$. In the continual leakage setting, we require an additional algorithm $\mathsf{SIG.Update}$ which updates the secret keys. Note that the verification key remains unchanged.

- $\mathsf{SIG.Gen}(1^\kappa) \to (\mathsf{vk}, \mathsf{sk}_0)$. The key generation algorithm takes in the security parameter κ, and outputs a secret key sk_0 and a public verification key vk.
- $\mathsf{SIG.Sign}(m, \mathsf{sk}_i) \to \sigma$. The signing algorithm takes in a message m and a secret key sk_i, and outputs a signature σ.
- $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma, m) \to \{0, 1\}$. The verification algorithm takes in the verification key vk, a signature σ, and a message m. It outputs either 0 or 1.
- $\mathsf{SIG.Update}(\mathsf{sk}_{i-1}) \to \mathsf{sk}_i$. The update algorithm takes in a secret key sk_{i-1} and produces a new secret key sk_i for the *same* verification key.

Correctness. The signature scheme satisfies correctness if $\mathsf{SIG.Verify}(\mathsf{vk}, \sigma, m)$ outputs 1 whenever $\mathsf{vk}, \mathsf{sk}_0$ is produced by $\mathsf{SIG.Gen}$, and σ is produced by $\mathsf{SIG.Sign}(m, \mathsf{sk}_i)$ for some sk_i obtained by calls to $\mathsf{SIG.Update}$, starting with sk_0. (If the verification algorithm is randomized, we may relax this requirement to hold with all but negligible probability.)

Security. We define continual leakage security for signatures in terms of the following game between a challenger and an attacker. (This extends the usual notion of existential unforgeability to our leakage setting.) The game is parameterized by two values: the security parameter κ, and the parameter μ which controls the amount of leakage allowed. For the sake of simplicity, we assume that the signing algorithm calls the update algorithm on each invocation. Since updates in our scheme do occur with each signature, we find it more convenient to work with the simplified definition given below.

Setup Phase The game begins with a setup phase. The challenger calls $\mathsf{Gen}(1^\kappa)$ to create the signing key, sk_0, and the verification key, vk. It gives vk to the attacker. No leakage is allowed in this phase.

Query Phase. In this phase, the attacker launches a polynomial number of signing queries and leakage queries. Each time, say in the ith query, the attacker specifies a message m_i and provides an efficiently computable leakage function f_i whose output is at most μ bits, and the challenger chooses randomness r_i, updates the secret key from sk_{i-1} to sk_i, and gives the attacker the corresponding signature for message m_i as well as the leakage response ℓ_i. In the CLR model, the leakage attack is applied on a single secret key, and the leakage response $\ell_i = f_i(\mathsf{sk}_{i-1})$. In the CLR with leakage on key updates, the leakage attack is applied on the current secret key and the randomness used for updating the secret key, i.e., $\ell_i = f_i(\mathsf{sk}_{i-1}, r_i)$.

Forgery Phase The attacker gives the challenger a message, m^*, and a signature σ^* such that m^* has not been previously queried. The attacker wins the game if (m^*, σ^*) passes the verification algorithm using vk.

Definition 3. (*Continual Leakage Resilience*) We say a Digital Signature scheme is μ-CLR secure (respectively, μ-CLR secure with leakage on key updates) if any PPT attacker only has a negligible advantage (negligible in κ) in the above game.

2.3. *Obfuscation*

Indistinguishability Obfuscation. A uniform PPT machine iO is called an indistinguishable obfucastor [4,5,29,33], for a circuit family $\{\mathcal{C}_\kappa\}$, if the following conditions hold:

– (Correctness) For all $\kappa \in \mathbb{N}$, for all $C \in \mathcal{C}_\kappa$, for all inputs x, we have

$$\Pr\left[C'(x) = C(x) \ : \ C' \leftarrow \mathsf{iO}(\kappa, C)\right] = 1$$

– For any uniform or non-uniform PPT distinguisher D, for all security parameter $\kappa \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\kappa$ such that $C_0(x) = C_1(x)$ for all inputs x, then

$$\left|\Pr\left[D(\mathsf{iO}(\kappa, C_0)) = 1\right] - \Pr\left[D(\mathsf{iO}(\kappa, C_1)) = 1\right]\right| \leq \mathsf{negl}(\kappa)$$

For simplicity, when the security parameter κ is clear, we write $\mathsf{iO}(C)$ in short.

Public-Coin Differing-inputs Obfuscation for Circuits. Barak et al. [4,5] defined the notion of differing-inputs obfuscation, which was later re-formulated in the works of Ananth et al. and Boyle et. al [3,9]. In our work, we use a weaker notion known as *public-coin differing-inputs obfuscation*, due to Ishai et al. [37]. To the best of our knowledge, unlike the case of differing-inputs obfuscation, there are no impossibility results for public-coin differing-inputs obfuscation. Below, we closely follow the definitions presented in [37].

Definition 4. (*Public-Coin Differing-Inputs Sampler for Circuits*) An efficient non-uniform sampling algorithm $\mathsf{Samp} = \{\mathsf{Samp}_\kappa\}$ is called a *public-coin differing-inputs sampler* for the parameterized collection of circuits $\mathcal{C} = \{\mathcal{C}_\kappa\}$ if the output of Samp_κ is distributed over $\mathcal{C}_\kappa \times \mathcal{C}_\kappa$ and for every efficient non-uniform algorithm $\mathcal{A} = \{\mathcal{A}_\kappa\}$ there exists negligible function negl such that for all $\kappa \in \mathbb{N}$:

$$\Pr_r[C_0(x) \neq C_1(x) : (C_0, C_1) \leftarrow \mathsf{Samp}_\kappa(r), x \leftarrow \mathcal{A}_\kappa(r)] \leq \mathsf{negl}(\kappa).$$

Note that in the above definition the sampler and attacker circuits both receive the same random coins as input.

Definition 5. (*Public-Coin Differing-inputs Obfuscator for Circuits*) A uniform PPT machine diO is called a public-coin differing-inputs obfuscator for the parameterized collection of circuits $\mathcal{C} = \{\mathcal{C}_\kappa\}$ if the following conditions are satisfied:

– (Correctness): For all security parameter κ, all $C \in \mathcal{C}_\kappa$, all inputs x, we have

$$\Pr[C'(x) = C(x) : C' \leftarrow \mathsf{diO}(\kappa, C)] = 1.$$

– (Differing-inputs): For every public-coin differing-inputs samplers $\mathsf{Samp} = \{\mathsf{Samp}_\kappa\}$ for the collection $\mathcal{C}$, for every (not necessarily uniform) PPT distinguisher D, there exists a negligible function negl such that for all security parameters κ:

$$\left|\begin{array}{l}\Pr[D_\kappa(r, C') = 1 : (C_0, C_1) \leftarrow \mathsf{Samp}_\kappa(r), C' \leftarrow \mathsf{diO}(\kappa, C_0)] - \\ \qquad \Pr[D_\kappa(r, C') = 1 : (C_0, C_1) \leftarrow \mathsf{Samp}_\kappa(r), C' \leftarrow \mathsf{diO}(\kappa, C_1)]\end{array}\right| \leq \mathsf{negl}(\kappa),$$

where the probability is taken over r and the coins of diO.

2.4. *Puncturable Pseudorandom Functions*

Puncturable family of PRFs are a special case of constrained PRFs [8,10,41], where the PRF is defined on all input strings except for a set of size polynomial in the security parameter. Below we recall their definition, as given by [47].

A puncturable family of PRFs PRF is defined by a tuple of efficient algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Punct})$ and a pair of polynomials $n()$ and $m()$:

– **Key Generation** $\mathsf{Gen}(1^\kappa)$ is a PPT algorithm that takes as input the security parameter κ and outputs a PRF key K.
– **Punctured Key Generation** $\mathsf{Punct}(K, S)$ is a PPT algorithm that takes as input a PRF key K, a set $S \subset \{0,1\}^{n(\kappa)}$ and outputs a punctured key K_S.
– **Evaluation** $\mathsf{Eval}(K, x)$ is a deterministic algorithm that takes as input a key K (punctured key or PRF key), a string $x \in \{0,1\}^{n(\kappa)}$ and outputs $y \in \{0,1\}^{m(\kappa)}$

Definition 6. A family of PRFs $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Punct})$ is puncturable if it satisfies the following properties

– **Functionality preserved under puncturing.** Let $K \leftarrow \mathsf{Gen}(1^\kappa)$ and $K_S \leftarrow \mathsf{Punct}(K, S)$. Then for all $x \notin S$, $\mathsf{Eval}(K, x) = \mathsf{Eval}(K_S, x)$.
– **Pseudorandom at (non-adaptively) punctured points.** For every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1()$ outputs a set $S \subset \{0,1\}^{n(\kappa)}$ and $x \in S$, consider an experiment $K \leftarrow \mathsf{Gen}(1^\kappa)$ and $K_S \leftarrow \mathsf{Punct}(K, S)$. Then

$$\left|\Pr[\mathcal{A}_2(K_S, x, \mathsf{Eval}(K, x)) = 1] - \Pr[\mathcal{A}_2(K_S, x, U_{m(\kappa)}) = 1]\right| \leq \mathsf{negl}(\kappa)$$

where $U_{m(\kappa)}$ denotes the uniform distribution over $m(\kappa)$ bits. Note that the set S is chosen non-adaptively, before the key K is generated.

Theorem 1. [8,10,32,41] *If one-way functions exist, then for all polynomial $n()$ and $m()$, there exists a puncturable PRF family that maps $n()$ bits to $m()$ bits.*

Next we consider families of PRFs that are with high probability injective:

Definition 7. A statistically injective (puncturable) PRF family with failure probability $\epsilon()$ is a family of (puncturable) PRFs such that with probability $1 - \epsilon(\kappa)$ over the random choice of key $K \leftarrow \mathsf{Gen}(1^\kappa)$, we have that $\mathsf{Eval}(K, \cdot)$ is injective.

If the failure probability function $\epsilon()$ is not specified, then $\epsilon()$ is a negligible function.

Theorem 2. [47] *If one-way functions exist, then for all efficiently computable functions $n(\kappa), m(\kappa)$, and $e(\kappa)$ such that $m(\kappa) > 2n(\kappa) + e(\kappa)$ here exists a puncturable statistically injective PRF family with failure probability $2^{-e(\kappa)}$ that maps $n(\kappa)$ bits to $m(\kappa)$ bits.*

Finally, we consider PRFs that are also (strong) extractors over their inputs:

Definition 8. An extracting (puncturable) PRF family with error $\epsilon()$ for min-entropy $k(\kappa)$ is a family of (puncturable) PRFs mapping $n(\kappa)$ bits to $m(\kappa)$ bits such that for all κ, if X is any distribution over $n(\kappa)$ bits with min-entropy greater than $k(\kappa)$ then the statistical distance between $(K \leftarrow \mathsf{Gen}(1^\kappa), \mathsf{Eval}(K, X))$ and $(K \leftarrow \mathsf{Gen}(1^\kappa), U_{m(\kappa)})$ is at most $\epsilon(\kappa)$, where U_ℓ denotes the uniform distribution over ℓ bit strings.

Theorem 3. [47] *If one-way functions exist, then for all efficiently computable functions $n(\kappa), m(\kappa), k(\kappa)$ and $e(\kappa)$ such that $n(\kappa) > k(\kappa) > m(\kappa) + 2e(\kappa) + 2$ there exists an extracting puncturable PRF family that maps $n(\kappa)$ bits to $m(\kappa)$ bits with error $2^{-e(\kappa)}$ for min-entropy $k(\kappa)$*

For ease of presentation, for a puncturable family of PRFs PRF, we often write $\mathsf{PRF}(K, x)$ to represent $\mathsf{PRF}.\mathsf{Eval}(K, x)$.

3. Compiler from 2CLR to Leakage on Key Updates

In this section, we present a compiler that upgrades any scheme for public key encryption (PKE) or digital signature (SIG), that is, consecutive two-key leakage resilient, into one that is secure against leakage on update. We first introduce a notion of *explainable update transformation*, which is a generalization of the idea of universal deniable encryption by Sahai and Waters [47]. We show how to use such a transformation to upgrade a scheme (PKE or SIG) that is secure in the consecutive two-key leakage model to one that is secure in the leak-on-update model (Sect. 3.2). Finally, we show two instantiations of the explainable update transformation: one based on indistinguishability obfuscation and the other on differing-inputs obfuscation (Sect. 3.3). For clarity of exposition, the following sections will focus on constructions of PKE. In Sect. 3.4, we show that the result can be translated to SIG.

3.1. *Consecutive Continual Leakage Resilience (2CLR)*

In this subsection, we present a new notion of *consecutive continual leakage resilience* for public key encryption (PKE). We remark that this notion can be easily extended to different cases, such as signatures or leakage-resilient one-way relations [21]. For simplicity and concreteness, we only present the PKE version. Let κ denote the security parameter and μ be the leakage bound between two updates. Let $\mathsf{PKE} = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update}\}$ be an encryption scheme with update.

Setup Phase. The game begins with a setup phase. The challenger calls $\mathsf{PKE.Gen}(1^\kappa)$ to create the initial secret key sk_0 and public key pk. It gives pk to the attacker. No leakage is allowed in this phase.

Query Phase. The attacker specifies an efficiently computable leakage function f_1, whose output is at most μ bits. The challenger updates the secret key (changing it from sk_0 to sk_1), and then gives the attacker $f_1(\mathsf{sk}_0, \mathsf{sk}_1)$. The attacker then repeats this a polynomial number of times, each time supplying an efficiently computable leakage function f_i whose output is at most μ bits. Each time, the challenger updates the secret key from sk_{i-1} to sk_i according to $\mathsf{Update}(\cdot)$ and gives the attacker $f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)$.

Challenge Phase. The attacker chooses two messages m_0, m_1 which it gives to the challenger. The challenger chooses a random bit $b \in \{0, 1\}$, encrypts m_b, and gives the resulting ciphertext to the attacker. The attacker then outputs a guess b' for b. The attacker wins the game if $b = b'$. We define the advantage of the attacker in this game as $|\frac{1}{2} - \Pr[b' = b]|$.

Definition 9. (*Continual Consecutive Leakage Resilience*) We say a public key encryption scheme is μ-leakage resilient against consecutive continual leakage (or μ-2CLR) if any probabilistic polynomial-time attacker only has a negligible advantage (negligible in κ) in the above game.

3.2. *Explainable Key Update Transformation*

Now we introduce a notion of *explainable key update transformation* and show how it can be used to upgrade security of a PKE scheme from 2CLR to CLR with leakage on key updates. Informally, an encryption scheme has an "explainable" update procedure if given both sk_{i-1} and $\mathsf{sk}_i = \mathsf{Update}(\mathsf{sk}_{i-1}, r_i)$, there is an efficient way to come up with some explained random coins $\hat{r}_i$ such that no adversary can distinguish the real coins r_i from the explained coins $\hat{r}_i$. Intuitively, this gives a way to handle leakage on random coins given just leakage on two consecutive keys.

We start with any encryption scheme PKE that has some key update procedure, and we introduce a transformation that produces a scheme PKE' with an *explainable* key update procedure.

Definition 10. (*Explainable Key Update Transformation*) Let $\mathsf{PKE} = \mathsf{PKE}.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update}\}$ be an encryption scheme with key update. An explainable key update transformation for PKE is a PPT algorithm $\mathsf{TransformGen}$ that takes input security parameter 1^κ, an update circuit C_{Update} (that implements the key update algorithm $\mathsf{PKE.Update}(1^\kappa, \cdot; \cdot)$), a public key pk of PKE, and outputs two programs $\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}$ with the following syntax:

Let $(\mathsf{pk}, \mathsf{sk})$ be a pair of public and secret keys of the encryption scheme

- $\mathcal{P}_{\mathsf{update}}$ takes inputs sk, random coins r, and $\mathcal{P}_{\mathsf{update}}(\mathsf{sk}; r)$ outputs an updated secret key sk';

– $\mathcal{P}_{\mathsf{explain}}$ takes inputs $(\mathsf{sk}, \mathsf{sk}')$, random coins $\bar{v}$, and $\mathcal{P}_{\mathsf{explain}}(\mathsf{sk}, \mathsf{sk}'; \bar{v})$ outputs a string r.

Given a polynomial $\rho(\cdot)$ and a public key pk, we define $\Pi_{\mathsf{pk}} = \bigcup_{j=0}^{\rho(\kappa)} \Pi_j$, where $\Pi_0 = \{\mathsf{sk} : (\mathsf{pk}, \mathsf{sk}) \in \mathsf{PKE.Gen}\}$, $\Pi_i = \{\mathsf{sk} : \exists \mathsf{sk}' \in \Pi_{i-1}, \mathsf{sk} \in \mathsf{Update}(\mathsf{sk}')\}$ for $i = 1, 2, \ldots, \rho(\kappa)$. In words, Π_{pk} is the set of all secret keys sk such that either $(\mathsf{pk}, \mathsf{sk})$ is in the support of $\mathsf{PKE.Gen}$ or sk can be obtained by the update procedure Update (up to polynomially many times) with an initial $(\mathsf{pk}, \mathsf{sk}') \in \mathsf{PKE.Gen}$.

We say the transformation is secure if:

(a) For any polynomial $\rho(\cdot)$, any pk, all $\mathsf{sk} \in \Pi_{\mathsf{pk}}$, any $\mathcal{P}_{\mathsf{update}} \in \mathsf{TransformGen}(1^\kappa, \mathsf{PKE.Update}, \mathsf{pk})$, the following two distributions are statistically close: $\{\mathcal{P}_{\mathsf{update}}(\mathsf{sk})\} \approx \{\mathsf{PKE.Update}(\mathsf{sk})\}$. Note that the circuit $\mathcal{P}_{\mathsf{update}}$ and the update algorithm $\mathsf{PKE.Update}$ might have different spaces for random coins, but the distributions can still be statistically close.

(b) For any public key pk and secret key $\mathsf{sk} \in \Pi_{\mathsf{pk}}$, the following two distributions are computationally indistinguishable:

$$\{(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}, \mathsf{sk}, u)\} \approx \{(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}, \mathsf{sk}, e)\},$$

where $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}) \leftarrow \mathsf{TransformGen}(1^\kappa, \mathsf{PKE.Update}, \mathsf{pk})$, $u \leftarrow U_{\mathrm{poly}(\kappa)}$, $\mathsf{sk}' = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}; u)$,
$e \leftarrow \mathcal{P}_{\mathsf{explain}}(\mathsf{sk}, \mathsf{sk}')$, and $U_{\mathrm{poly}(\kappa)}$ denotes the uniform distribution over a polynomial number of bits.

Let $\mathsf{PKE} = \mathsf{PKE}.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update}\}$ be a public key encryption scheme and $\mathsf{TransformGen}$ be an explainable key update transformation for PKE as above. We define the following transformed scheme $\mathsf{PKE}' = \mathsf{PKE}'.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update}\}$ as follows:

– $\mathsf{PKE}'.\mathsf{Gen}(1^\kappa)$: compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$.
Then compute $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}) \leftarrow \mathsf{TransformGen}(1^\kappa, \mathsf{PKE.Update}, \mathsf{pk})$.
Finally, output $\mathsf{pk}' = (\mathsf{pk}, \mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}})$ and $\mathsf{sk}' = \mathsf{sk}$.
– $\mathsf{PKE}'.\mathsf{Enc}(\mathsf{pk}', m)$: parse $\mathsf{pk}' = (\mathsf{pk}, \mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}})$. Then output $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m)$.
– $\mathsf{PKE}'.\mathsf{Dec}(\mathsf{sk}', c)$: output $m = \mathsf{PKE.Dec}(\mathsf{sk}', c)$.
– $\mathsf{PKE}'.\mathsf{Update}(\mathsf{sk}')$: output $\mathsf{sk}'' \leftarrow \mathcal{P}_{\mathsf{update}}(\mathsf{sk}')$.

Then we are able to show the following theorem for the upgraded scheme PKE'.

Theorem 4. *Let* $\mathsf{PKE} = \mathsf{PKE}.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update}\}$ *be a public key encryption scheme that is* μ*-2CLR (without leakage on update), and* $\mathsf{TransformGen}$ *a secure explainable key update transformation for* PKE. *Then the transformed scheme* $\mathsf{PKE}' = \mathsf{PKE}'.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update}\}$ *described above is* μ*-CLR with leakage on key updates.*

Proof. Assume toward contradiction that there is a PPT adversary $\mathcal{A}$ and a non-negligible $\epsilon(\cdot)$ such that for infinitely many values of κ, $\mathsf{Adv}_{\mathcal{A},\mathsf{PKE}'} \geq \epsilon(\kappa)$ in the leak-

on-update model. Then we show that there exists $\mathcal{B}$ that breaks the security of the underlying PKE (in the consecutive two-key leakage model) with probability $\epsilon(\kappa) - \mathsf{negl}(\kappa)$. This is a contradiction.

For notional simplicity, we will use $\mathsf{Adv}_{\mathcal{A},\mathsf{PKE}'}$ to denote the advantage of the adversary $\mathcal{A}$ attacking the scheme PKE' (according to leak-on-update attacks), and $\mathsf{Adv}_{\mathcal{B},\mathsf{PKE}}$ to denote the advantage of the adversary $\mathcal{B}$ attacking the scheme PKE (according to consecutive two-key leakage attacks).

We define $\mathcal{B}$ in the following way: $\mathcal{B}$ internally instantiates $\mathcal{A}$ and participates externally in a continual consecutive two-key leakage experiment on public key encryption scheme PKE'. Specifically, $\mathcal{B}$ does the following:

- Upon receiving pk^* externally, $\mathcal{B}$ runs $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}) \leftarrow \mathsf{TransformGen}(1^\kappa, \mathsf{PKE.Update}, \mathsf{pk}^*)$. Note that by the properties of the transformation, this can be done given only pk^*. $\mathcal{B}$ sets $\mathsf{pk}' = (\mathsf{pk}^*, \mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}})$ to be the public key for the PKE' scheme and forwards pk' to $\mathcal{A}$.
- When $\mathcal{A}$ asks for a leakage query $f(\mathsf{sk}'_{i-1}, r_i)$, $\mathcal{B}$ asks for the following leakage query on $(\mathsf{sk}_{i-1}, \mathsf{sk}_i)$: $f'(\mathsf{sk}_{i-1}, \mathsf{sk}_i) = f(\mathsf{sk}_{i-1}, \mathcal{P}_{\mathsf{explain}}(\mathsf{sk}_{i-1}, \mathsf{sk}_i))$ and forwards the response to $\mathcal{A}$. Note that the output lengths of f and f' are the same.
- At some point, $\mathcal{A}$ submits m_0, m_1 and $\mathcal{B}$ forwards them to its external experiment.
- Upon receiving the challenge ciphertext c^*, $\mathcal{B}$ forwards it to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

Now we would like to analyze the advantage of $\mathcal{B}$. It is easy to see that $\mathcal{B}$ has the same advantage as $\mathcal{A}$; however, there is a subtlety such that $\mathcal{A}$ does not necessarily have advantage $\epsilon(\kappa)$: The simulation of leakage queries provided by $\mathcal{B}$ is not identical to the distribution in the real game that $\mathcal{A}$ would expect. Recall that in the security experiment of the scheme PKE', the secret keys are updated according to $\mathcal{P}_{\mathsf{update}}$. In the above experiment (where $\mathcal{B}$ set up), the secret keys were updated using the Update externally, and the random coins were simulated by the $\mathcal{P}_{\mathsf{explain}}$ algorithm.

Our goal is to show that actually $\mathcal{A}$ has essentially the same advantage in this modified experiment as in the original experiment. We show this by the following lemma:

Lemma 1. *For any polynomial n, the following two distributions are computationally indistinguishable.*

$$D_1 \equiv (\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}, \mathsf{sk}_0, r_1, \mathsf{sk}_1, \ldots, \mathsf{sk}_{n-1}, r_n, \mathsf{sk}_n) \approx$$
$$D_2 \equiv (\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}, \mathsf{sk}_0, \widehat{r}_1, \widehat{\mathsf{sk}}_1, \ldots, \widehat{\mathsf{sk}}_{n-1}, \widehat{r}_n, \widehat{\mathsf{sk}}_n),$$

where the initial $\mathsf{pk}, \mathsf{sk}_0$ *and* $\mathsf{TransformGen}(1^\kappa, \mathsf{pk})$ *are sampled identically in both experiment; in* D_1, $\mathsf{sk}_{i+1} = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}_i; r_{i+1})$, *and* r_{i+1}*'s are uniformly random; in* D_2, $\widehat{\mathsf{sk}}_{i+1} \leftarrow \mathsf{Update}(\widehat{\mathsf{sk}}_i), \widehat{r}_{i+1} \leftarrow \mathcal{P}_{\mathsf{explain}}(\widehat{\mathsf{sk}}_i, \widehat{\mathsf{sk}}_{i+1})$. *(Note* $\widehat{\mathsf{sk}}_0 = \mathsf{sk}_0$.*)*

Proof. To show the lemma, we consider the following hybrids: for $i \in [n]$ define

$$H^{(i)} = (\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}, \mathsf{sk}_0, \widehat{r}_1, \widehat{\mathsf{sk}}_1, \ldots, \widehat{\mathsf{sk}}_{i-1}, r_i, \mathsf{sk}_i, r_{i+1}, \mathsf{sk}_{i+1}, r_{i+2}, \ldots, \mathsf{sk}_n),$$

where the experiment is identical to D_2 for up to $\widehat{\mathsf{sk}}_{i-1}$. Then it samples a uniformly random r_i, sets $\mathsf{sk}_i = \mathcal{P}_{\mathsf{update}}(\widehat{\mathsf{sk}}_{i-1}; r_i)$, and proceeds as D_1.

$$H^{(i.5)} = (\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}, \mathsf{sk}_0, \widehat{r}_1, \widehat{\mathsf{sk}}_1, \ldots, \widehat{\mathsf{sk}}_{i-1}, \widehat{r}_i, \mathsf{sk}_i, r_{i+1}, \mathsf{sk}_{i+1}, r_{i+2}, \ldots, \mathsf{sk}_n),$$

where the experiment is identical to $H^{(i)}$ for up to $\widehat{\mathsf{sk}}_{i-1}$, and then, it samples $\mathsf{sk}_i \leftarrow \mathcal{P}_{\mathsf{update}}(\widehat{\mathsf{sk}}_{i-1})$, and $\widehat{r}_i \leftarrow \mathcal{P}_{\mathsf{explain}}(\widehat{\mathsf{sk}}_{i-1}, \mathsf{sk}_i)$. The experiment is identical to D_1 for the rest.

Then we establish the following lemmas, and the lemma follows directly.

Lemma 2. *For $i \in [n-1]$, $H^{(i.5)}$ is statistically close to $H^{(i+1)}$.*

Lemma 3. *For $i \in [n]$, $H^{(i)}$ is computationally indistinguishable from $H^{(i.5)}$.*

This first lemma follows directly from the property (a) of Definition 10. We now prove Lemma 3.

Proof. Suppose there exists a (poly-sized) distinguisher $\mathcal{D}$ that distinguishes $H^{(i)}$ from $H^{(i.5)}$ with non-negligible probability, then there exist pk^*, sk^*, and another $\mathcal{D}'$ that can break the property (b).

From the definition of the experiments, we know that $\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}$ are independent of the public key and the first i secret keys, i.e., $\boldsymbol{p} = (\mathsf{pk}, \mathsf{sk}_0, \widehat{\mathsf{sk}}_1, \ldots, \widehat{\mathsf{sk}}_{i-1})$. By an average argument, there exists a fixed

$$\boldsymbol{p}^* = (\mathsf{pk}^*, \mathsf{sk}_0^*, \widehat{\mathsf{sk}}_1^*, \ldots, \widehat{\mathsf{sk}}_{i-1}^*)$$

such that $\mathcal{D}$ can distinguish $H^{(i)}$ from $H^{(i.5)}$ conditioned on $\boldsymbol{p}^*$ with non-negligible probability. (The probability is over the randomness of the rest experiment.) Then we are going to argue that there exist a poly-sized distinguisher $\mathcal{D}'$, a key pair $\mathsf{pk}', \mathsf{sk}'$ such that $\mathcal{D}'$ can distinguish $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}', \mathsf{sk}', u)$ from $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}', \mathsf{sk}', e)$ where u is from the uniform distribution, $\mathsf{sk}'' = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}'; u)$, and $e \leftarrow \mathcal{P}_{\mathsf{explain}}(\mathsf{sk}', \mathsf{sk}'')$.

Let $\mathsf{pk}' = \mathsf{pk}^*$, $\mathsf{sk}' = \widehat{\mathsf{sk}}_{i-1}^*$, and we define $\mathcal{D}'$ (with the prefix $\boldsymbol{p}^*$ hardwired) who on the challenge input $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}', \mathsf{sk}', z)$ does the following:

- For $j \in [i-1]$, $\mathcal{D}'$ samples $\widehat{r}_j = \mathcal{P}_{\mathsf{explain}}(\mathsf{sk}_{j-1}^*, \mathsf{sk}_j^*)$.
- Set $\mathsf{sk}_{i-1} = \mathsf{sk}'$ and $r_i = z$, $\mathsf{sk}_i = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}_{i-1}, z)$.
- For $j \geq i+1$, $\mathcal{D}'$ samples r_j from the uniform distribution and sets $\mathsf{sk}_j = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}_{j-1}; r_j)$.
- Finally, $\mathcal{D}'$ outputs $\mathcal{D}(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}', \mathsf{sk}_0^*, \widehat{r}_1, \mathsf{sk}_1^*, \ldots, \mathsf{sk}_{i-1}, r_i, \mathsf{sk}_i, r_{i+1}, \ldots, \mathsf{sk}_n)$.

Clearly, if the challenge z was sampled according to uniformly random (as u), then $\mathcal{D}'$ will output according to $\mathcal{D}(H^{(i)}|_{\boldsymbol{p}*})$. On the other hand, suppose it was sampled according to $\mathcal{P}_{\mathsf{explain}}$ (as e), then $\mathcal{D}'$ will output according to $\mathcal{D}(H^{i.5}|_{\boldsymbol{p}^*})$. This completes the proof of the lemma. □

Remark 1. The non-uniform argument above is not necessary. We present in this way for simplicity. The uniform reduction can be obtained using a standard Markov type argument, which we omit here.

Now, we are ready to analyze the advantage of $\mathcal{B}$ (and $\mathcal{A}$). Denote $\mathsf{Adv}_{\mathcal{A},\mathsf{PKE}';D}$ as the advantage of $\mathcal{A}$ in the experiment where the leakage queries are answered according to the distribution D. By assumption, we know that $\mathsf{Adv}_{\mathcal{A},\mathsf{PKE}';D_1} = \epsilon(\kappa)$, and by definition the leakage queries are answered according to D_1. By the above lemma, we know that $|\mathsf{Adv}_{\mathcal{A},\mathsf{PKE}';D_1} - \mathsf{Adv}_{\mathcal{A},\mathsf{PKE}';D_2}| \leq \mathsf{negl}(\kappa)$; otherwise, D_1 and D_2 are distinguishable. Thus, we know $\mathsf{Adv}_{\mathcal{A},\mathsf{PKE}';D_2} \geq \epsilon(\kappa) - \mathsf{negl}(\kappa)$. It is not hard to see that $\mathsf{Adv}_{\mathcal{B},\mathsf{PKE}} = \mathsf{Adv}_{\mathcal{A},\mathsf{PKE}';D_2}$, since $\mathcal{B}$ answers $\mathcal{A}$'s the leakage queries exactly according the distribution D_2. Thus, $\mathsf{Adv}_{\mathcal{B},\mathsf{PKE}} \geq \epsilon(\kappa) - \mathsf{negl}(\kappa)$, which is a contradiction. This completes the proof of the theorem. □

3.3. *Instantiations via Obfuscation*

In this section, we show how to build an explainable key update transformation from program obfuscation. There are two variants of our construction: one from the weaker notion of indistinguishability obfuscation (iO) [5,29] and one from the stronger notion of public-coin differing-inputs obfuscation (public-coin diO) [37]. Since our best parameters are achieved using public-coin diO, we present the public-coin diO variant of our construction/proof and indicate the points in the construction/proof where the iO variant differs.

Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update})$ be a public key encryption scheme (or a signature scheme with algorithms $\mathsf{Verify}, \mathsf{Sign}$) with key update, and diO(resp. iO) be a public-coin differing-inputs obfuscator (resp. indistinguishability obfuscator) for some class defined later. Let κ be a security parameter. Let L_{sk} be the length of secret keys in PKE and L_r be the length of randomness used by Update. For ease of notation, we suppress the dependence of these lengths on κ. We note that in the 2CLR case, it is without loss of generality to assume $L_r << L_{\mathsf{sk}}$, because we can always use pseudorandom coins (e.g., the output of a PRG) to do the update. Since only the two consecutive keys are leaked (not the randomness, e.g., the seed to the PRG), the update with the pseudorandom coins remains secure, assuming the PRG is secure.

Let $\mathcal{H}$ be a family of public-coin collision-resistant hash functions, as well as a family of $(2\kappa, \epsilon)$-good extractor,[9] mapping $2L_{\mathsf{sk}} + 2\kappa$ bits to κ bits. Let F_1 and F_2 be families of puncturable pseudorandom functions, where F_1 has input length $2L_{\mathsf{sk}} + 3\kappa$ bits and

[9] The description of the hash function is the seed. On input seed and source, the extractor outputs a distribution that is ϵ close to the uniform distribution if the source has min-entropy 2κ. Here we set ϵ to be some negligible. The hash function is chosen from a family of functions, and once chosen, it is a deterministic function.

Internal (hardcoded) state: Public key pk, keys K_1, K_2, and h.

On input secret key sk_1; randomness $u = (u_1, u_2)$.

- If $F_2(K_2, u_1) \oplus u_2 = (\mathsf{sk}_2, r')$ for (proper length) strings sk_2, r' and $u_1 = h(\mathsf{sk}_1, \mathsf{sk}_2, r')$, then output sk_2.
- Else let $x = F_1(K_1, (\mathsf{sk}_1, u))$. Output $\mathsf{sk}_2 = \mathsf{PKE.Update}(\mathsf{pk}, \mathsf{sk}_1; x)$.

Fig. 1. Program update.

Internal (hardcoded) state: key K_2.

On input secret keys $\mathsf{sk}_1, \mathsf{sk}_2$; randomness $r \in \{0,1\}^\kappa$

- Set $u_1 = h(\mathsf{sk}_1, \mathsf{sk}_2, r)$. Set $u_2 = F_2(K_2, u_1) \oplus (\mathsf{sk}_2, r)$. Output $e = (u_1, u_2)$.

Fig. 2. Program explain.

output length L_r bits, and it is as well an $(L_r + \kappa, \epsilon)$-good unseeded extractor; F_2 has input length κ and output length $L_{\mathsf{sk}}+2\kappa$. Here $|u_1| = \kappa$ and $|u_2| = L_{\mathsf{sk}}+2\kappa$, $|r'| = 2\kappa$.

Define the algorithm $\mathsf{TransformGen}(1^\kappa, \mathsf{pk})$ that on input the security parameter, a public key pk and a circuit that implements $\mathsf{PKE.Update}(\cdot)$ as follows:

- $\mathsf{TransformGen}$ samples K_1, K_2 as keys for the puncturable PRF as above, and $h \leftarrow \mathcal{H}$. Let P_1 be the program as Fig. 1, and P_2 as Fig. 2.
- Then it samples $\mathcal{P}_{\mathsf{update}} \leftarrow \mathsf{diO}(P_1)$, and $\mathcal{P}_{\mathsf{explain}} \leftarrow \mathsf{diO}(P_2)$. It outputs $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}})$.

The iO *variant.* Essentially, the difference between the construction based on iO versus public-coin diO is that the hash function $\mathcal{H}$ is replaced with an injective, puncturable PRF, $F_3 : \{0,1\}^\kappa \times \{0,1\}^{2L_{\mathsf{sk}}+\kappa} \to \{0,1\}^{4L_{\mathsf{sk}}+3\kappa}$, which can be constructed from OWF (see [47]). The iO-based construction is a simplified version of the deniable encryption of the work [47], where our construction does not use a PRG in the Explain program. The security proof relies directly on the puncturing technique on the key K_3 with a condition check. We elaborate on the details below.

- Instead of sampling a hash function $h \leftarrow \mathcal{H}$, $\mathsf{TransformGen}$ samples an additional PRF key $K_3 \leftarrow F.\mathsf{Gen}(1^\kappa)$.
- We modify program P_1 in Fig. 1 by embedding an additional key, K_3, and checking whether $u_1 = F_3(K_3, \mathsf{sk}_1, \mathsf{sk}_2, r')$.
- We modify program P_2 in Fig. 2, by embedding an additional key, K_3, and setting $u_1 = F_3(K_3, \mathsf{sk}_1, \mathsf{sk}_2, r)$.
- All input/output lengths of the programs and pseudorandom functions are adjusted to be consistent with the fact that u_1 now has length $4L_{\mathsf{sk}} + 3\kappa$ (whereas previously it had length κ).

We now establish the following theorem.

Theorem 5. *Let* PKE *be any public key encryption scheme with key update. Assume* diO *(resp.* iO*) is a secure* public-coin *differing-inputs indistinguishable obfuscator*

(resp. indistinguishable obfuscator) for the circuits required by the construction, F_1, F_2 are puncturable pseudorandom functions as above, and $\mathcal{H}$ is a family of public-coin collision-resistant hash functions as above. Then the transformation $\mathsf{TransformGen}$ *(resp.* $\mathsf{TransformGen}'$*) defined above is a secure explainable update transformation for* PKE *as defined in Definition 10, which takes randomness $u = (u_1, u_2)$ of length $L_1 + L_2$, where $L_1 := \kappa$, $L_2 := L_{\mathsf{sk}} + 2\kappa$ (resp. $L_1 := 4L_{\mathsf{sk}} + 3\kappa$, $L_2 := L_{\mathsf{sk}} + 2\kappa$).*

Looking at the big picture, recall that the entire secret state required for continually updating the secret key consists of the current secret state, sk, and randomness, $u = (u_1, u_2)$, which together have total length $L_{\mathsf{sk}} + L_1 + L_2$. Thus, when plugging in our public-coin diO-based construction, we ultimately achieve leakage rate of $\frac{\mu}{2L_{\mathsf{sk}}+2\kappa} = \frac{\mu}{2L_{\mathsf{sk}}} - o(1)$, where μ is the leakage rate of the underlying 2CLR public key encryption scheme. On the other hand, when plugging in our iO-based construction, we achieve leakage rate of $\frac{\mu}{6L_{\mathsf{sk}}+5\kappa} = \frac{\mu}{6L_{\mathsf{sk}}} - o(1)$.

Proof. Recall that to show that $\mathsf{TransformGen}$ satisfies property (a) of Definition 10, we need to demonstrate that for any polynomial $\rho(\cdot)$, any pk, all $\mathsf{sk} \in \Pi_{\mathsf{pk}}$, any $\mathcal{P}_{\mathsf{update}} \in \mathsf{TransformGen}(1^\kappa, \mathsf{PKE.Update}, \mathsf{pk})$, the following two distributions are statistically close: $\{\mathcal{P}_{\mathsf{update}}(\mathsf{sk})\} \approx \{\mathsf{PKE.Update}(\mathsf{sk})\}$. Inspecting program $\mathcal{P}_{\mathsf{update}}(\mathsf{sk})$, the above follows in a straightforward manner from the following: (1) When u is chosen uniformly at random, the probability that $F_2(K_2, u_1) \oplus u_2 = (\mathsf{sk}_2, r')$ and $u_1 = h(\mathsf{sk}_1, \mathsf{sk}_2, r')$ is negligible and (2) when u is chosen uniformly at random, then $x = F_1(K_1, (\mathsf{sk}_1, u))$ is statistically close to uniform. For the analysis showing that (1) holds, see the analysis of Hybrid 1. (2) holds since F_1 is an $(L_r + \kappa, \epsilon)$-good unseeded extractor.

Recall that to show that $\mathsf{TransformGen}$ satisfies property (b) of Definition 10, we need to demonstrate that for any public key pk^* and secret key $\mathsf{sk}^* \in \Pi_{\mathsf{pk}}$, the following two distributions are computationally indistinguishable:

$$\{(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}^*, \mathsf{sk}^*, u^*)\} \approx \{(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}^*, \mathsf{sk}^*, e^*)\},$$

where these values are generated by

1. $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}) \leftarrow \mathsf{TransformGen}(1^\kappa, \mathsf{PKE.Update}, \mathsf{pk}^*)$,
2. $u^* = (u_1^*, u_2^*) \leftarrow \{0,1\}^{L_{\mathsf{sk}}+3\kappa}$ (in the iO variant, $u^* = (u_1^*, u_2^*) \leftarrow \{0,1\}^{3L_{\mathsf{sk}}+4\kappa}$),
3. Set $x^* = F_1(K_1, \mathsf{sk}^* || u^*)$, $\mathsf{sk}' = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}^*; u^*)$. Then choose uniformly random r^* of length κ, and set $e_1^* = h(\mathsf{sk}^*, \mathsf{sk}', r^*)$ (in the iO variant, $e_1^* = F_3(K_3, \mathsf{sk}^*, \mathsf{sk}', r^*)$) and $e_2^* = F_2(K_2, e_1^*) \oplus (\mathsf{sk}', r^*)$.

We prove this through the following sequence of hybrid steps.

Hybrid 1: In this hybrid step, we change Step 3 of the above challenge. Instead of computing $\mathsf{sk}' = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}^*; u^*)$, we compute $\mathsf{sk}' = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}^*; x^*)$:

1. $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}) \leftarrow \mathsf{TransformGen}(1^\kappa, \mathsf{PKE.Update}, \mathsf{pk}^*)$,
2. $u^* = (u_1^*, u_2^*) \leftarrow \{0,1\}^{L_{\mathsf{sk}}+3\kappa}$ (in the iO variant, $u^* = (u_1^*, u_2^*) \leftarrow \{0,1\}^{3L_{\mathsf{sk}}+4\kappa}$),
3. Set $x^* = F_1(K_1, \mathsf{sk}^* || u^*)$, $\mathsf{sk}' = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}^*; x^*)$, and choose uniformly random r^* of length κ. Then, $e_1^* = h(\mathsf{sk}^*, \mathsf{sk}', r^*)$ (in the iO variant, $e_1^* = F_3(K_3, \mathsf{sk}^*, \mathsf{sk}', r^*)$) and $e_2^* = F_2(K_2, e_1^*) \oplus (\mathsf{sk}', r^*)$.

Note that the only time in which this changes the experiment is when the values $(u_1^*, u_2^*) \leftarrow \{0,1\}^{L_{\mathsf{sk}}+3\kappa}$ happen to satisfy $F_2(K_2, u_1^*) \oplus u_2^* = (\mathsf{sk}', r')$ such that $u_1^* = h(\mathsf{sk}^*, \mathsf{sk}', r')$. For any fixed $u_1^*, \mathsf{sk}^*, \mathsf{sk}'$, and a random u_{2^*}, we know the marginal probability of r' is still uniform given $u_1^*, \mathsf{sk}^*, \mathsf{sk}'$. Therefore, we have $\Pr_{u_{2^*}}[h(\mathsf{sk}^*, \mathsf{sk}', r') = u_1^*] = \Pr_{r'}[h(\mathsf{sk}^*, \mathsf{sk}', r') = u_1^*] < 2^{-\kappa} + \epsilon$. This is because h is a $(2\kappa, \epsilon)$-extractor, so the output of h is ϵ-close to uniform over $\{0,1\}^\kappa$, and a uniform distribution hits a particular string with probability $2^{-\kappa}$. Since we set ϵ to be some negligible, the two distributions are only different with the negligible quantity.
The iO *variant.* We make the same modification as in the public-coin diO case and set $\mathsf{sk}' = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}^*; x^*)$. However, the analysis of the hybrid changes, as we now describe: In the iO setting, the only time in which the above modification changes the output of the experiment is when the values $(u_1^*, u_2^*) \leftarrow \{0,1\}^{3L_{\mathsf{sk}}+4\kappa}$ happen to satisfy $F_2(K_2, u_1^*) \oplus u_2^* = (\mathsf{sk}', r')$ such that $u_1^* = F_3(K_3, \mathsf{sk}^*, \mathsf{sk}', r')$. The only way the above can be satisfied is if u_1^* is in the range of $F_3(K_3, \cdot)$. Note that the range of $F_3(K_3, \cdot)$ has size $2^{4L_{\mathsf{sk}}+3\kappa}$, while u_1^* is chosen independently and uniformly at random from a domain of size $2L_{\mathsf{sk}} + 2\kappa$. This means that the probability that u_1^* is in the range of $F_3(K_3, \cdot)$ is at most $\frac{2^{2L_{\mathsf{sk}}+\kappa}}{2^{4L_{\mathsf{sk}}+3\kappa}} = 2^{-2L_{\mathsf{sk}}-2\kappa}$, which is negligible.

Hybrid 2: In this hybrid step, we modify the program in Fig. 1, puncturing key K_1 at points $\{\mathsf{sk}^*||u^*\}$ and $\{\mathsf{sk}^*||e^*\}$, and adding a line of code at the beginning of the program to ensure that the PRF is never evaluated at these two points. See Fig. 3. We claim that with overwhelming probability over the choice of u^*, this modified program has identical input/output as the program that was used in Hybrid 1 (Fig. 1). Note that on input (sk^*, e^*) the output of the original program was already sk' as defined in Hybrid 1, so the outputs of the two programs are identical on this input. (This follows because e^* anyway encodes sk', so when the "If"-statement is triggered in the program of Fig. 1, the output is sk'.) As long as u_1^* and u_2^* do not have the property that $F_2(K_2, u_1^*) \oplus u_2^* = (\mathsf{sk}', r')$ such that $u_1^* = h(\mathsf{sk}^*, \mathsf{sk}', r')$, then the programs have identical output on input (sk^*, u^*) as well. (This follows because sk' is defined as $\mathsf{sk}' = \mathcal{P}_{\mathsf{update}}(\mathsf{sk}^*; F_1(K_1, \mathsf{sk}^*||u^*))$ in the challenge game, which is also the output of the program in Fig. 1 when u_1^* and u_2^* fail this condition.) As we argued in Hybrid 1, with very high probability, u^* does not have this property. (We stress that u^* is fixed *before* we construct the obfuscated program described in Fig. 3, so with overwhelming probability over the choice of u^*, the two programs have identical input output behavior.) Indistinguishability of Hybrids 1 and 2 follows from the security of the obfuscation. Note that this hybrid requires only the weaker notion of *indistinguishability obfuscation*.
The iO *variant.* The modification and security argument are identical, with the exception that we require that u_1^* and u_2^* do not have the property that $F_2(K_2, u_1^*) \oplus u_2^* = (\mathsf{sk}', r')$ such that $u_1^* = F_3(K_3, \mathsf{sk}^*, \mathsf{sk}', r')$. We invoke the argument from the previous hybrid to show that this is true with overwhelming probability over choice of u^*.

Hybrid 3: In this hybrid, we change the challenge game to use truly random x^* when computing $\mathsf{sk}' = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}^*; x^*)$ (instead of $x^* = F_1(K_1; \mathsf{sk}^*||u^*)$). Security holds by a reduction to the pseudorandomness of F_1 at the punctured point (sk^*, u^*). More specifically, given an adversary $\mathcal{A}$ that distinguishes Hybrid 2 from Hybrid 3 on values $\mathsf{pk}^*, \mathsf{sk}^*$, we describe an reduction $\mathcal{B}$ that attacks the security of the

Internal (hardcoded) state: Public key pk^*, keys $\widetilde{K}_1 = \mathsf{PRF.Punct}(K_1, \{\mathsf{sk}^*||u^*\}, \{\mathsf{sk}^*||e^*\})$, K_2, sk' (as defined in Hybrid 1) and h.

On input secret key sk_1; randomness $u = (u_1, u_2)$.

- If $(\mathsf{sk}_1, u) = (\mathsf{sk}^*, u^*)$ or $(\mathsf{sk}_1, u) = (\mathsf{sk}^*, e^*)$ output the value sk'.
- Else If $F_2(K_2; u_1) \oplus u_2 = (\mathsf{sk}_2, r')$ such that $u_1 = h(\mathsf{sk}_1, \mathsf{sk}_2, r')$, then output sk_2.
- Else let $x = F_1(K_1, \mathsf{sk}_1||u)$. Output $\mathsf{sk}_2 = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}_1; x)$.

Fig. 3. Program update, as used in Hybrid 2.

Internal (hardcoded) state: Public key pk^*, keys $\widetilde{K}_1 = \mathsf{PRF.Punct}(K_1, \{\mathsf{sk}^*||u^*\}, \{\mathsf{sk}^*||e^*\})$, $\widetilde{K}_2 = \mathsf{PRF.Punct}(K_2, \{u_1^*\}, \{e_1^*\})$, sk' (as defined in Hybrid 3) and h.

On input secret key sk_1; randomness $u = (u_1, u_2)$.

- If $(\mathsf{sk}_1, u) = (\mathsf{sk}^*, u^*)$ or $(\mathsf{sk}_1, u) = (\mathsf{sk}^*, e^*)$ output value sk'.
- Else If $u_1 = u_1^*$ or $u_1 = e_1^*$, let $x = F_1(\widetilde{K}_1, \mathsf{sk}_1||u)$. Output $\mathsf{sk}_2 = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}_1; x)$.
- Else
  - If $F_2(K_2; u_1) \oplus u_2 = (\mathsf{sk}_2, r')$ such that $u_1 = h(\mathsf{sk}_1, \mathsf{sk}_2, r')$, then output sk_2.
  - Else let $x = F_1(\widetilde{K}_1, \mathsf{sk}_1||u)$. Output $\mathsf{sk}_2 = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}_1; x)$.

Fig. 4. Program update, as used in Hybrid 4.

puncturable PRF, F_1. $\mathcal{B}$ generates u^* at random and submits (sk^*, u^*) to his challenger. He receives $\widetilde{K}_1 = \mathsf{PRF.Punct}(K_1, \{\mathsf{sk}^*||u^*\})$, and a value x^* as a challenge. $\mathcal{B}$ computes $\mathsf{sk}' = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}^*; x^*)$, chooses r^* at random, and computes e^* as in the original challenge game. He creates $\mathcal{P}_{\mathsf{update}}$ using $\widetilde{K}_1$ and sampling K_2 honestly. The same K_2 is used for creating $\mathcal{P}_{\mathsf{explain}}$. $\mathcal{B}$ obfuscates both circuits, which completes the simulation of $\mathcal{A}$'s view.

The iO *variant.* The modification and security argument are identical for the iO setting.

Hybrid 4: In this hybrid, we puncture K_2 at both u_1^* and e_1^*, and modify the Update program to output appropriate hardcoded values on these inputs. (See Fig. 4.) To prove that Hybrids 3 and 4 are indistinguishable, we rely on security of public-coin differing-inputs obfuscation and public-coin collision-resistant hash function. In particular, we will show that suppose the hybrids are distinguishable, then we can break the security of the collision-resistant hash function.

Consider the following sampler $\mathsf{Samp}(1^\kappa)$: outputs C_0, C_1 as the two update programs as in Hybrids 3 and 4, respectively, and it outputs an auxiliary input $\mathsf{aux} = (\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{sk}', u^*, e^*, K_2, h, r^*)$ sampled as in the both hybrids. Note that aux includes all the random coins of the sampler. Suppose there exists a distinguisher $\mathcal{D}$ for the two hybrids, then there exists a distinguished $\mathcal{D}'$ that distinguishes $(\mathsf{diO}(C_0), \mathsf{aux})$ from $(\mathsf{diO}(C_1), \mathsf{aux})$. This is because given the challenge input, $\mathcal{D}'$ can complete the rest of the experiment either according to Hybrid 3 or Hybrid 4. Then by security of the diO, we know there exists an adversary (extractor) $\mathcal{B}$ that given (C_0, C_1, aux) finds an input

such that C_0 and C_1 evaluate differently. However, this contradicts the security of the public-coin collision-resistant hash function. We establish this by the following lemma.

Lemma 4. *Assume h is sampled from a family of public-coin collision-resistant hash function, (and $(2\kappa, \epsilon)$-extracting) as above. Then for any* PPT *adversary, the probability is negligible to find a differing-inputs given (C_0, C_1, aux) as above.*

Proof. By examining the two circuits, we observe that the differing-inputs have the following two forms: $(\bar{\mathsf{sk}}, u_1^*, \bar{u}_2)$ such that $u_1^* = h(\bar{\mathsf{sk}}, F_2(K_2; u_1^*) \oplus \bar{u}_2)$, $(\bar{\mathsf{sk}}, \bar{u}_2) \neq (\mathsf{sk}^*, u_2^*)$; or $(\bar{\mathsf{sk}}, e_1^*, \bar{e}_2)$ such that $e_1^* = h(\bar{\mathsf{sk}}, F_2(K_2; e_1^*) \oplus \bar{e}_2)$, $(\bar{\mathsf{sk}}, \bar{e}_2) \neq (\mathsf{sk}^*, e_2^*)$. This is because they will run enter the first Else IF in Hybrid 3 (Fig. 3), but enter the modified line (the first Else IF) in Hybrid 4 (Fig. 4). We argue that both cases happen with negligible probability; otherwise, security of the hash function can be broken.

For the first case, we observe that the collision resistance and $(2\kappa, \epsilon)$ extracting guarantee that the probability of finding an pre-image of a random value u_1^* is small, even given aux; otherwise, there is an adversary who can break collision resistance. For the second case, we know that $e_1^* = h(\mathsf{sk}^*, \mathsf{sk}', r^*) = h(\bar{\mathsf{sk}}, F_2(K_2; e_1^*) \oplus \bar{e}_2) = h(\bar{\mathsf{sk}}, e_2^* \oplus (\mathsf{sk}', r^*) \oplus \bar{e}_2)$. Since we know that $(\bar{\mathsf{sk}}, \bar{e}_2) \neq (\mathsf{sk}^*, e_2^*)$, we find a collision, which again remains hard even given aux.

Thus, suppose there exists a differing-inputs finder $\mathcal{A}$, we can define an adversary $\mathcal{B}$ to break the collision-resistant hash function: On input h, $\mathcal{B}$ simulates the sampler Samp with the h. Then it runs $\mathcal{A}$ to find a differing-inputs. Then according to the above argument, either of the two cases will lead to finding a collision. □

The iO *variant.* The hybrid proceeds identically to the diO variant. To prove that Hybrids 3 and 4 are indistinguishable, we rely on the security of the indistinguishability obfuscator. In particular, we will show that the functionality of program update is identical in the two hybrids. By examining the two circuits, we must show that if the event $(u_1 = u_1^* \vee u_1 = e_1^*) \wedge F_2(k_2; u_1) \oplus u_2 = (\mathsf{sk}_2, r')$ such that $u_1 = F_3(K_3, \mathsf{sk}_1, \mathsf{sk}_2, r')$ occurs then $\mathsf{sk}_2 = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}_1; x)$, where $x = F_1(\widetilde{K}_1, \mathsf{sk}_1 || u)$. Indeed, this is the only case where we enter the first Else IF in Hybrid 3 (Fig. 3), but enter the modified line (the first Else IF) in Hybrid 4 (Fig. 4). To see that the above holds, we consider two cases: $u_1 = u_1^*$ and $u_1 = e_1^*$. First, if $u_1 = u_1^*$ then, as argued above, u_1 is not in the range of $F_3(K_3, \cdot)$ (with overwhelming probability over the choice of u_1^*) and so the event cannot occur. Second, note that if $u_1 = e_1^*$ then $u_1 = F_3(K_3, \mathsf{sk}^*, \mathsf{sk}', r^*)$. Since $F_3(K_3, \cdot)$ is injective, if the above event occurs, it means that $\mathsf{sk}_1 = \mathsf{sk}^*$ and $u_2 = (\mathsf{sk}', r^*) \oplus F_2(k_2; u_1) = e_2^*$. This in turn means that $x = F_1(\widetilde{K}_1, \mathsf{sk}^* || e^*) = x^*$. Therefore, by definition of sk', we have that $\mathsf{sk}_2 = \mathsf{sk}' = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}^*; x^*) = \mathsf{PKE.Update}(\mathsf{pk}^*, \mathsf{sk}_1; x)$, as desired.

Hybrid 5: In this hybrid, we puncture K_2 at both u_1^* and e_1^*, and modify the Explain program to output appropriate hardcoded values on these inputs. (See Fig. 5.) Similar to the argument for the previous hybrids, we argue that Hybrids 4 and 5 are indistinguishable by security of the public-coin differing-inputs obfuscation and public-coin collision-resistant hash function. Consider a sampler $\mathsf{Samp}(1^\kappa)$: outputs C_0, C_1 as the two explain programs as in Hybrids 4 and 5, respectively, and it outputs an auxiliary input $\mathsf{aux} =$

Internal (hardcoded) state: key $\widetilde{K}_2 = \mathsf{PRF.Punct}(K_2, \{u_1^*\}, \{e_1^*\}), u^*, e^*$.

On input secret keys $\mathsf{sk}_1, \mathsf{sk}_2$; randomness $r \in \{0,1\}^\kappa$

- If $u_1^* = h(\mathsf{sk}_1, \mathsf{sk}_2, r)$, output u^*. Else If $e_1^* = h(\mathsf{sk}_1, \mathsf{sk}_2, r)$, output e^*.
- Else, set $u_1 = h(\mathsf{sk}_1, \mathsf{sk}_2, r)$. Set $u_2 = F_2(K_2, u_1) \oplus (\mathsf{sk}_2, r)$. Output $e = (u_1, u_2)$.

Fig. 5. Program explain, as used in Hybrid 5.

$(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{sk}', u^*, e^*, K_2, h, r^*)$ sampled as in the both hybrids (note that aux includes all the random coins of the sampler). Similar to the above argument, suppose there exists a distinguisher $\mathcal{D}$ that distinguishes Hybrids 4 and 5, then we can construct a distinguisher $\mathcal{D}'$ that distinguishes $(\mathsf{diO}(C_0), \mathsf{aux})$ from $(\mathsf{diO}(C_1), \mathsf{aux})$. This is because given the challenging input, $\mathcal{D}'$ can simulate the hybrids. Then by security of the diO, there exists an adversary (extractor) $\mathcal{B}$ that can find differing-inputs. Now we want to argue that suppose the h comes from a public-coin collision-resistant hash family, then no PPT adversary can find differing-inputs. This leads to a contradiction.

Lemma 5. *Assume h is sampled from a family of public-coin collision-resistant hash function, (and $(2\kappa, \epsilon)$-extracting) as above. Then for any* PPT *adversary, the probability is negligible to find a differing-inputs given (C_0, C_1, aux) as above.*

Proof. The proof is almost identical to that of Lemma 4. We omit the details. □

The iO *variant.* Program explain is modified with the following line: If $u_1^* = F_3(K_3, \mathsf{sk}_1, \mathsf{sk}_2, r)$, output u^*. Else If $e_1^* = F_3(K_3, \mathsf{sk}_1, \mathsf{sk}_2, r)$, output e^*. The proof is almost identical to that of the previous hybrid.

Hybrid 6: In this hybrid, we change both e_1^* and e_2^* to uniformly random. Hybrids 5 and 6 are indistinguishable by the security of the puncturable PRF F_2, and by the fact that h is $(2\kappa, \epsilon)$-extracting. Clearly in this hybrid, the distributions of $\{(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}^*, \mathsf{sk}^*, u^*)\}$ and $\{(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}, \mathsf{pk}^*, \mathsf{sk}^*, e^*)\}$ are identical. From the indistinguishable arguments that the original game and Hybrid 6 are indistinguishable, we can argue that the distributions in the original game are indistinguishable. This concludes the proof.

The iO *variant.* We must first puncture K_3 at $(\mathsf{sk}^*, \mathsf{sk}', r^*)$. and modify both Update and Explain so that whenever we check whether $u_1 = F_3(K_3, \mathsf{sk}^*, \mathsf{sk}', r^*)$, we instead check whether $u_1 = e_1^*$. Once we have done this, we can now proceed as in the diO variant and switch both e_1^* and e_2^* to uniformly random. □

3.4. *Extension to Digital Signatures*

We have already demonstrated a compiler that upgrades any 2CLR public key encryption scheme into one that is secure against leakage on update. The same results can be translated to digital signature scheme.

Continual Consecutive Leakage Resilience for Digital Signature. Following the presentation in Subsect. 3.1, we can modify the security game and define continual consecutive leakage resilience for digital signature schemes. We say a digital signature scheme is μ-leakage resilient against consecutive continual leakage (or μ-2CLR) if any probabilistic polynomial-time attacker only has a negligible advantage (negligible in κ) in the modified game.

Explainable Key-Update Transformation for Digital Signature Scheme. Following the presentation in Subsect. 3.2, we can define explainable key update transformation for digital signature scheme. We start with any digital signature scheme $\mathsf{SIG} = \mathsf{SIG}.\{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update}\}$ that has some key update procedure. Then we can follow Definition 10, and introduce a transformation $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}) \leftarrow \mathsf{TransformGen}(1^\kappa, \mathsf{SIG.Update}, \mathsf{pk})$. This transformation will help us define a scheme $\mathsf{SIG}' = \mathsf{SIG}'.\{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update}\}$ with an explainable key update procedure.

- $\mathsf{SIG}'.\mathsf{Gen}(1^\kappa)$: compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Gen}(1^\kappa)$.
  Then compute $(\mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}}) \leftarrow \mathsf{TransformGen}(1^\kappa, \mathsf{SIG.Update}, \mathsf{pk})$.
  Finally, output $\mathsf{pk}' = (\mathsf{pk}, \mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}})$ and $\mathsf{sk}' = \mathsf{sk}$.
- $\mathsf{SIG}'.\mathsf{Sign}(\mathsf{sk}', m)$: output $\sigma = \mathsf{SIG.Sign}(\mathsf{sk}', m)$.
- $\mathsf{SIG}'.\mathsf{Verify}(\mathsf{pk}', m, \sigma)$: parse $\mathsf{pk}' = (\mathsf{pk}, \mathcal{P}_{\mathsf{update}}, \mathcal{P}_{\mathsf{explain}})$. Then output $b \leftarrow \mathsf{SIG.Verify}(\mathsf{pk}, m, \sigma)$.
- $\mathsf{SIG}'.\mathsf{Update}(\mathsf{sk}')$: output $\mathsf{sk}'' \leftarrow \mathcal{P}_{\mathsf{update}}(\mathsf{sk}')$.

Similarly, we can show a theorem for the upgraded scheme SIG'.

Theorem 6. *Let* $\mathsf{SIG} = \mathsf{SIG}.\{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update}\}$ *be a digital signature scheme that is μ-2CLR (without leakage on update), and* $\mathsf{TransformGen}$ *a secure explainable key update transformation for* SIG*. Then the transformed scheme* $\mathsf{SIG}' = \mathsf{SIG}'.\{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update}\}$ *described above is μ-CLR with leakage on key updates.*

The poof of the above theorem can exactly follow the proof of Theorem 4 and so we omit the proof.

Instantiations. Finally, as in Subsect. 3.3, we can instantiate the explainable update transformation via indistinguishability obfuscation or differing-inputs obfuscation, and establish the same theorem for digital signature schemes.

Theorem 7. *Let* SIG *be any digital signature scheme with key update. Assume* diO *(resp.* iO*) is a secure* public-coin *differing-inputs indistinguishable obfuscator (resp. indistinguishable obfuscator) for the circuits required by the construction, F_1, F_2 are puncturable pseudorandom functions as above, and $\mathcal{H}$ is a family of public-coin collision-resistant hash functions as above. Then the transformation* $\mathsf{TransformGen}$ *(resp.* $\mathsf{TransformGen}'$*) defined above is a secure explainable update transformation for* SIG *which takes randomness $u = (u_1, u_2)$ of length $L_1 + L_2$, where $L_1 := \kappa, L_2 := L_{\mathsf{sk}} + 2\kappa$ (resp. $L_1 := 4L_{\mathsf{sk}} + 3\kappa, L_2 := L_{\mathsf{sk}} + 2\kappa$).*

4. 2CLR from "Leakage-Resilient Subspaces"

We show that the PKE scheme of Brakerski et al. [12] (BKKV), which has been proved CLR, can achieve 2CLR (with a slight adjustment in the scheme's parameters). We note that our focus on PKE here is justified by the fact that we show generically that any CLR (resp. 2CLR) PKE scheme implies a CLR "one-way relation" (OWR) [21]; to the best of our knowledge, such an implication was not previously known. Therefore, by the results of Dodis et al. [21], this translates all our results about PKE to the signature setting as well. We show that the approach of Dodis et al. [21] for constructing CLR OWRs can be extended to 2CLR one-way relations, but we achieve weaker parameters this way.

Recall that in the work [12], to prove that their scheme is CLR, they show *"random subspaces are leakage resilient"*. In particular, they show that for a random subspace X, the statistical difference between $\big(X, f(v)\big)$ and $\big(X, f(u)\big)$ is negligible, where f is an arbitrary length-bounded function, v is a random point in the subspace, and u is a random point in the whole space. Then by a simple hybrid argument, they show that $\big(X, f_1(v_0), f_2(v_1), \ldots, f_t(v_{t-1})\big)$ and $\big(X, f_1(u_0), f_2(u_1), \ldots, f_t(u_{t-1})\big)$ are indistinguishable, where $f_1, \ldots, f_t$ are arbitrary and adaptively chosen length-bounded functions, $v_0, v_1, \ldots, v_{t-1}$ are independent random points in the subspace, and $u_0, u_1, \ldots, u_{t-1}$ are independent random points in the whole space. This lemma plays the core role in their proof.

In order to show that their scheme satisfies the 2CLR security, we consider random subspaces under "consecutive" leakage. That is, we want to show:

$$\begin{aligned}&\big(X, f_1(v_0, v_1), f_2(v_1, v_2), \ldots, f_t(v_{t-1}, v_t)\big)\\ &\quad\approx \big(X, f_1(u_0, u_1), f_2(u_1, u_2), \ldots, f_t(u_{t-1}, u_t)\big),\end{aligned}$$

for arbitrary and adaptively chosen f_i's, i.e., each f_i can be chosen after seeing the previous leakage values $f_1, \ldots, f_{i-1}$. However, this does not follow by a hybrid argument of $\big(X, f(v)\big) \approx \big(X, f(u)\big)$, because in the 2CLR case each point is leaked twice. It is not clear how to embed a challenging instance of $(X, f(z))$ into the larger experiment while still being able to simulate the rest.

To handle this technical issue, we establish a new lemma showing *random subspaces are "consecutive" leakage resilient*. With the lemma and a hybrid argument, we can show that the above experiments are indistinguishable. Then we show how to use this fact to prove that the scheme of BKKV is 2CLR.

Lemma 6. *Let $t, n, \ell, d \in \mathbb{N}$, $n \geq \ell \geq 3d$, and q be a prime. Let $(A, X) \leftarrow \mathbb{Z}_q^{t\times n} \times \mathbb{Z}_q^{n\times \ell}$ such that $A \cdot X = 0$, $T, T' \leftarrow \mathsf{Rk}_d(\mathbb{Z}_q^{\ell\times d})$, $U \leftarrow \mathbb{Z}_q^{n\times d}$ such that $A \cdot U = 0$, (i.e., U is a random matrix in $\mathsf{Ker}(A)$), and $f : \mathbb{Z}_q^{t\times n} \times \mathbb{Z}_q^{n\times 2d} \to W$ be any function.*[10] *Then we have:*

$$\Delta\Big(\big(A, X, f(A, XT, XT'), XT'\big), \big(A, X, f\left(A, U, XT'\right), XT'\big)\Big) \leq \epsilon,$$

[10] Note: Rk denotes rank. Here we use n as the dimension (different from [12] who used m) to avoid overloading notation.

as long as $|W| \leq (1 - 1/q) \cdot q^{\ell - 3d + 1} \cdot \epsilon^2$.

Proof. We will actually prove something stronger, namely we will prove, under the assumptions of Lemma 6, that

$$\Delta\left(\left(A, X, f(A, X \cdot T, X \cdot T'), X \cdot T', T'\right), \left(A, X, f(A, U, X \cdot T'), X \cdot T', T'\right)\right) \leq \frac{1}{2}\sqrt{\frac{3|W|}{(1 - 1/q)q^{\ell - 3d + 1}}} < \epsilon \ .$$

Note that this implies the lemma by solving for ϵ, after noting that ignoring the last component in each tuple can only decrease statistical difference.

For the proof, we will apply Lemma 7 as follows. We will take hash function H to be $H \colon \mathbb{Z}_q^{n \times \ell} \times \mathbb{Z}_q^{\ell \times d} \to \mathbb{Z}_q^{n \times d}$ where $H_K(D) = KD$ (matrix multiplication), and take the set $\mathcal{Z}$ to be $\mathbb{Z}_q^{n \times \ell} \times \mathbb{Z}_q^{\ell \times d}$. Next we take random variable K to be uniform on $\mathbb{Z}_q^{n \times \ell}$ (denoted as the matrix X), D to be uniform on $\mathsf{Rk}_d(\mathbb{Z}_q^{\ell \times d})$, and finally $Z = (A, XT', T')$ where A is uniform conditioned on $AX = 0$, $T' \in \mathsf{Rk}_d(\mathbb{Z}_q^{\ell \times d})$ is independent uniform. We define $U_{|Z}$ as the uniform distribution such that $AU = 0$. This also means that U is a random matrix in the kernel of A.

It remains to prove under these settings that

$$\Pr\left[\, (D, D', Z) \in \mathsf{BAD} \,\right] \leq \frac{1}{(1 - 1/q)q^{\ell - 3d + 1}}$$

with BAD defined as in Lemma 7. For this, let us consider

$$\Delta\big((H_{K|Z}(T_1), H_{K|Z}(T_2)), (U_{|Z}, U'_{|Z})\big)$$

where $Z = (A, XT', T')$ as defined above. The above statistical distance is zero as long as the outcomes of T_1, T_2, T' are all linearly independent. This is so because $\ell \geq 3d$. Now, by a standard formula the probability that T_1, T_2, T' have a linear dependency is bounded by $\frac{1}{(1-1/q)q^{\ell - 3d + 1}}$, and we are done. □

We note that this lemma is slightly different that the original lemma in the work [12]: the leakage function considered here also takes in a public matrix A, which is used as the public key in the system. We observe that both our work and [12] need this version of the lemma to prove security of the encryption scheme.

We actually prove Lemma 6 as a consequence of a new generalization of the Crooked Leftover Hash Lemma (LHL) [6,26] we introduce (to handle hash functions that are only pairwise independent if some bad event does not happen), as follows.

Lemma 7. *Let* $H \colon \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ *be a hash function and* (K, Z) *be joint random variables over* $(\mathcal{K}, \mathcal{Z})$ *for the set* $\mathcal{K}$ *and some set* $\mathcal{Z}$. *Define the following set*

$$\mathsf{BAD} = \Big\{(d, d', z) \in \mathcal{D} \times \mathcal{D} \times \mathcal{Z} : \Delta\big((H_{K|_{Z=z}}(d), H_{K|_{Z=z}}(d')), (U_{|Z=z}, U'_{|Z=z})\big) > 0\Big\}, \tag{1}$$

where $U_{|Z=z}, U'_{|Z=z}$ denote two independent uniform distributions over $\mathcal{R}$ conditioned on $Z = z$, and $K|_{Z=z}$ is the conditional distribution of K given $Z = z$. We note that $\mathcal{R}$ might depend on z, so when we describe a uniform distribution over $\mathcal{R}$, we need to specify the condition $Z = z$.

Suppose D and D' are i.i.d. random variables over $\mathcal{D}$, (K, Z) are random variables over $\mathcal{K} \times \mathcal{Z}$ satisfying $\Pr\big[(D, D', Z) \in \mathsf{BAD}\big] \leq \epsilon'$. Then for any set $\mathcal{S}$ and function $f : \mathcal{R} \times \mathcal{Z} \to \mathcal{S}$ it holds that

$$\Delta((K, Z, f(H_K(D), Z)), (K, Z, f(U_{|Z}, Z))) \leq \frac{1}{2}\sqrt{3\epsilon' \, |\mathcal{S}|}\,.$$

Proof. The proof is an extension of the proof of the Crooked LHL given in [6]. First, using Cauchy–Schwarz and Jensen's inequality we have

$$\Delta((K, Z, f(H_K(D), Z)), (K, Z, f(U_{|Z}, Z))) \leqq \frac{1}{2}\sqrt{|S| \, \mathbf{E}_{k,z}\left[\sum_s (\Pr[\, f(H_k(D), z) = s\,] - \Pr\big[\, f(U_{|Z=z}, z) = s\,\big])^2\right]},$$

where $U_{|Z=z}$ is uniform on $\mathcal{R}$ conditioned on $Z = z$, and the expectation is over (k, z) drawn from (K, Z). Thus, to complete the proof it suffices to prove the following lemma.

Lemma 8.

$$\mathbf{E}_{k,z}\left[\sum_s \Big(\Pr[\, f(H_k(D), z) = s\,] - \Pr\big[\, f(U_{|Z=z}, z) = s\,\big]\Big)^2\right] \leq 3\epsilon'\,. \tag{2}$$

Proof. By the linearity of expectation, we can express Eq. 2 as:

$$\begin{aligned}&\mathbf{E}_{k,z} \sum_s \Pr[\, f(H_k(D), z) = s\,]^2 \\ &-2\mathbf{E}_{k,z} \sum_s \Pr[\, f(H_k(D), z) = s\,]\Pr\big[\, f(U_{|Z=z}, z) = s\,\big] \\ &+\mathbf{E}_z \mathsf{Col}(f(U_{|Z=z}, z)),\end{aligned} \tag{3}$$

where $U_{|Z=z}$ is uniform on $\mathcal{R}$ conditioned on $Z = z$, and Col is the collision probability of its input random variable. Note that since $f(U_{|Z=z}, z)$ is independent of k, we can drop it in the third term. In the following, we are going to calculate bounds for the first two terms.

For any $s \in \mathcal{S}$, we can write $\Pr[\, f(H_k(D), z) = s \,] = \sum_d \Pr[\, D = d \,] \delta_{f(H_k(d),z),s}$ where $\delta_{a,b}$ is 1 if $a = b$ and 0 otherwise, and thus

$$\sum_s \Pr[\, f(H_k(D), z) = s \,]^2 = \sum_{d,d'} \Pr[\, D = d \,] \Pr\left[\, D = d' \,\right] \delta_{f(H_k(d),z),\, f(H_k(d'),z)} \,.$$

So we have

$$\begin{aligned}
&\mathbf{E}_{k,z} \sum_s \Pr[\, f(H_k(D), z) = s \,]^2 \\
&= \mathbf{E}_{k,z} \left[\sum_{d,d'} \Pr[\, D = d \,] \Pr\left[\, D = d' \,\right] \delta_{f(H_k(d),z),\, f(H_k(d'),z)} \right] \\
&= \mathbf{E}_z \left[\sum_{d,d'} \Pr[\, D = d \,] \Pr\left[\, D = d' \,\right] \mathbf{E}_k \left[\delta_{f(H_k(d),z),\, f(H_k(d'),z)} \right] \right] \\
&\leq \sum_{z,d,d' \notin \mathsf{BAD}} \Pr[\, Z = z \,] \Pr[\, D = d \,] \Pr\left[\, D = d' \,\right] \mathbf{E}_k \left[\delta_{f(H_k(d),z),\, f(H_k(d'),z)} \right] + \epsilon' \\
&= \mathbf{E}_z \left[\mathsf{Col}(f(U_{|Z=z}, z)) \right] + \epsilon', \qquad (4)
\end{aligned}$$

where BAD is defined as in Eq. (1) from Lemma 7. The inequality holds because, by our definition of BAD, if $(z, d, d') \notin \mathsf{BAD}$, $(H_k(d), H_k(d'))$ are distributed exactly as two uniformly chosen elements (conditioned on $Z = z$), and because $\Pr[(z, d, d') \in \mathsf{BAD}] \leq \epsilon'$.

By a similar calculation, we have:

$$\begin{aligned}
&\mathbf{E}_{k,z} \sum_s \Pr[\, f(H_k(D), z) = s \,] \Pr\left[\, f(U_{|Z=z}, z) = s \,\right] \\
&\quad \geq \mathbf{E}_z \left[\mathsf{Col}(f(U_{|Z=z}, z)) \right] - \epsilon' \,. \qquad (5)
\end{aligned}$$

For the same reason, $H_k(D)$ is uniformly random except for the bad event, whose probability is bounded by ϵ'.

Putting things together, the inequality in Eq. 2 follows immediately by plugging the bounds in Eqs. 4 and 5. This concludes the proof. □

Here we describe the BKKV encryption scheme and show it is 2CLR secure. We begin by presenting the main scheme in BKKV, which uses the weaker linear assumption in bilinear groups, but achieves a worse leakage rate (that can tolerate roughly $1/8 \cdot |\mathsf{sk}| - o(\kappa)$). In that work [12], it is also pointed out that under the stronger SXDH assumption in bilinear groups, the rate can be improved to tolerate roughly $1/4 \cdot |\mathsf{sk}| - o(k)$, with essentially the same proof. The same argument also holds in the 2CLR setting. To avoid repetition, we just describe the original scheme in BKKV and prove that it is actually 2CLR under the linear assumption in bilinear groups.

- **Parameters.** Let G, G_T be two groups of prime order p such that there exists a bilinear map $e : G \times G \to G_T$. Let g be a generator of G (and so $e(g, g)$ is a generator of G_T). An additional parameter $\ell \geq 7$ is polynomial in the security parameter. (Setting different ℓ will enable a trade-off between efficiency and the rate of tolerable leakage). For the scheme to be secure, we require that the linear assumption holds in the group G, which implies that the size of the group must be super-polynomial, i.e., $p = \kappa^{\omega(1)}$.
- **Key generation.** The algorithm samples $A \leftarrow \mathbb{Z}_p^{2\times\ell}$, and $Y \leftarrow \mathsf{Ker}^2(A)$, i.e., $Y \in \mathbb{Z}_p^{\ell\times 2}$ can be viewed as two random (linearly independent) points in the kernel of A. Then it sets $\mathsf{pk} = g^A$, $\mathsf{sk} = g^Y$. Note that since A is known, Y can be sampled efficiently.
- **Key update.** Given a secret key $g^Y \in G^{\ell\times 2}$, the algorithm samples $R \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{2\times 2})$ and then sets $\mathsf{sk}' = g^{Y\cdot R}$.
- **Encryption.** Given a public key $\mathsf{pk} = g^A$, to encrypt 0, it samples a random $r \in \mathbb{Z}_p^2$ and outputs $c = g^{r^T\cdot A}$. To encrypt 1, it just outputs $c = g^{u^T}$ where $u \leftarrow \mathbb{Z}_p^\ell$ is a uniformly random vector.
- **Decryption.** Given a ciphertext $c = g^{v^T}$ and a secret key $\mathsf{sk} = g^Y$, the algorithm computes $e(g, g)^{v^T\cdot Y}$. If the result is $e(g, g)^0$, then it outputs 0, otherwise 1.

Then we are able to achieve the following theorem:

Theorem 8. *Under the linear assumption, for every $\ell \geq 7$, the encryption scheme above is μ-bit leakage resilient against two-key continual and consecutive leakage, where $\mu = \frac{(\ell-6)\cdot\log p}{2} - \omega(\kappa)$. Note that the leakage rate would be $\frac{\mu}{|\mathsf{sk}|+|\mathsf{sk}|} \approx 1/8$, as ℓ is chosen sufficiently large.*

Proof. The theorem follows directly from the following lemma:

Lemma 9. *For any $t \in \mathrm{poly}(\kappa), r \leftarrow \mathbb{Z}_p^2, A \leftarrow \mathbb{Z}_p^{2\times\ell}$, random $Y \in \mathsf{Ker}^2(A)$, and polynomial sized functions $f_1, f_2, \ldots, f_t$ where each $f_i : \mathbb{Z}_p^{\ell\times 2} \times \mathbb{Z}_p^{\ell\times 2} \to \{0,1\}^\mu$ and can be adaptively chosen (i.e., f_i can be chosen after seeing the leakage values of $f_1, \ldots, f_{i-1}$), the following two distributions, D_0 and D_1, are computationally indistinguishable:*

$$D_0 = (g, g^A, g^{r^T\cdot A}, f_1(\mathsf{sk}_0, \mathsf{sk}_1), \ldots f_t(\mathsf{sk}_{t-1}, \mathsf{sk}_t))$$
$$D_1 = (g, g^A, g^u, f_1(\mathsf{sk}_0, \mathsf{sk}_1), \ldots f_t(\mathsf{sk}_{t-1}, \mathsf{sk}_t)),$$

where $\mathsf{sk}_0 = g^Y$ and sk_{i+1} is the updated key from sk_i using random coins $R_i \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{2\times 2})$ as defined in the key update procedure.

Basically, the distribution D_0 is the view of the adversary when given an encryption of 0 as the challenge ciphertext and continual leakage of the secret keys; D_1 is the same except the challenge ciphertext is an encryption of 1. Our goal is to show that no polynomial sized adversary can distinguish between them.

We show the lemma in the following steps:

1. We first consider two modified experiment D'_0 and D'_1 where in these experiments, all the secret keys are sampled independently, i.e., $\mathsf{sk}'_{i+1} \leftarrow \mathsf{Ker}^2(A)$. In other words, instead of using a rotation of the current secret key, the update procedure re-samples two random (linearly independent) points in the kernel of A. Denote $D'_b = (g, g^A, g^z, f_1(\mathsf{sk}'_0, \mathsf{sk}'_1), \ldots f_t(\mathsf{sk}'_{t-1}, \mathsf{sk}'_t))$ for g^z is sampled either from $g^{r^T \cdot A}$ or g^u depending on $b \in \{0, 1\}$. Intuitively, the operations are computed in the exponent, so the adversary cannot distinguish between the modified experiments from the original ones. We formally prove this using the linear assumption.
2. Then we consider the following modified experiments: for $b \in \{0, 1\}$, define

$$D''_b = (g, g^A, g^z, f_1(g^{u_0}, g^{u_1}), f_2(g^{u_1}, g^{u_2}), \cdots, f_t(g^{u_{t-1}}, g^{u_t})),$$

   where the distribution samples a random $X \in \mathbb{Z}_p^{\ell \times (\ell-3)}$ such that $A \cdot X = 0$; then it samples each $u_i = X \cdot T_i$ for $T_i \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{(\ell-3)\times 2})$; finally, it samples z either as $r^T \cdot A$ or uniformly random as in D'_b. We then show that D''_b is indistinguishable from D'_b using the new geometric lemma.
3. Finally, we show that $D''_0 \approx D''_1$ under the linear assumption.

To implement the approach just described, we establish the following lemmas.

Lemma 10. *For $b \in \{0, 1\}$, D_b is computationally indistinguishable from D'_b.*

To show this lemma, we first establish a lemma:

Lemma 11. *Under the linear assumption, $(g, g^A, g^Y, g^{Y \cdot U}) \approx (g, g^A, g^Y, g^{Y'})$, where $A \leftarrow \mathbb{Z}_p^{2\times \ell}$, $Y, Y' \leftarrow \mathsf{Ker}^2(A)$, and $U \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{2\times 2})$.*

Suppose there exists a distinguisher $\mathcal{A}$ that breaks the above statement with non-negligible probability, then we can construct $\mathcal{B}$ that can break the linear assumption (the matrix form). In particular, $\mathcal{B}$ distinguishes $(g, g^C, g^{C \cdot U})$ from $(g, g^C, g^{C'})$ where C and C' are two independent and uniformly random samples from $\mathbb{Z}_p^{(\ell-2)\times 2}$, and U is uniformly random matrix from $\mathbb{Z}_p^{2\times 2}$. Note that when $p = \kappa^{\omega(1)}$ (this is required by the linear assumption), then with overwhelming probability, $(C||C')$ is a rank 4 matrix, and $(C||C \cdot U)$ is a rank 2 matrix. The linear assumption is that no polynomial-time adversary can distinguish the two distributions when given in the exponent.

$\mathcal{B}$ does the following on input (g, g^C, g^Z), where Z is either $C \cdot U$ or a uniformly random matrix C':

- $\mathcal{B}$ samples a random rank 2 matrix $A \in \mathbb{Z}_p^{2\times \ell}$. Then $\mathcal{B}$ computes an arbitrary basis of $\mathsf{Ker}(A)$ (note that $\mathsf{Ker}(A) = \{v \in \mathbb{Z}_p^\ell : A \cdot v = 0\}$), denoted as X. By the rank-nullity theorem (see any linear algebra textbook), the dimension of $\mathsf{Ker}(A)$ plus $\mathsf{Rk}(A)$ is ℓ. So we know that $X \in \mathbb{Z}_p^{\ell \times (\ell-2)}$, i.e., X contains $(\ell - 2)$ vectors that are linearly independent.
- $\mathcal{B}$ computes $g^{X \cdot C}$ and $g^{X \cdot Z}$. This can be done efficiently given (g^C, g^Z) and X in the clear.
- $\mathcal{B}$ outputs $\mathcal{A}(g, g^A, g^{X \cdot C}, g^{X \cdot Z})$.

We observe that when $p = \kappa^{\omega(1)}$, the distribution of A is statistically close to a random matrix, and U is statistically close to a random rank 2 matrix. Then it is not hard to see that $g^{X \cdot C}$ is identically distributed to g^{Y}, and $g^{X \cdot Z}$ is distributed as $g^{(X \cdot C) \cdot U}$ if $Z = C \cdot U$, and otherwise as $g^{Y'}$. So $\mathcal{B}$ can break the linear assumption with probability essentially the same as that of $\mathcal{A}$. This completes the proof of the lemma.

Then Lemma 10 can be proved using the lemma via a standard hybrid argument. We show that $D_0 \approx D'_0$ and the other one can be shown by the same argument. For $i \in [t+1]$, define hybrids H_i as the experiment as D_0 except the first i secret keys are sampled independently, as D'_0; the rest are sampled according to rotations, as D_0. It is not hard to see that $H_1 = D_0$, $H_{t+1} = D'_0$, and $H_i \approx H_{i+1}$ using the lemma. The argument is obvious and standard, so we omit the detail.

Then we recall the modified distribution D''_b: for $b \in \{0, 1\}$,

$$D''_b = (g, g^A, g^z, f_1(g^{u_0}, g^{u_1}), f_2(g^{u_1}, g^{u_2}), \cdots, f_t(g^{u_{t-1}}, g^{u_t})),$$

where the distribution samples a random $X \in \mathbb{Z}_p^{\ell \times (\ell-2)}$ such that $A \cdot X = 0$; then it samples each $u_i = X \cdot T_i$ for $T_i \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{(\ell-2)\times 2})$, and z is sampled either $r^T \cdot A$ or uniformly random. We then establish the following lemma.

Lemma 12. *For $b \in \{0, 1\}$, D'_b is computationally indistinguishable from D''_b.*

We prove the lemma using another hybrid argument. We prove that $D'_0 \approx D''_0$, and the other follows from the same argument. We define hybrids Q_i for $i \in [t]$ where in Q_i, the first i secret keys (the exponents) are sampled randomly from $\mathsf{Ker}^2(A)$ (as D'_0), and the rest secret keys (the exponents) are sampled as $X \cdot T$ (as D''_0). Clearly, $Q_0 = D''_0$ and $Q_{t+1} = D'_0$. Then we want to show that Q_i is indistinguishable from Q_{i+1} using the extended geometric lemma (Lemma 6).

For any $i \in [t+1]$, we argue that suppose there exists an (even unbounded) adversary that distinguishes Q_i from Q_{i+1} with probability better than ϵ, then there exist a leakage function L and an adversary $\mathcal{B}$ such that $\mathcal{B}$ can distinguish $\left(A, X, L(A, X \cdot T, X \cdot T'), X \cdot T'\right)$ from $\left(A, X, L(A, U, X \cdot T'), X \cdot T'\right)$ in Lemma 6 with probability better than $\epsilon - \mathsf{negl}(\kappa)$ (dimensions will be set later). We will set the parameters of Lemma 6 such that the two distributions have negligible statistical difference; thus ϵ can be at most a negligible quantity.

Now we formally set the dimensions: let X be a random matrix in $\mathbb{Z}^{\ell \times (\ell-3)}$; T, T' be two random rank 2 matrices in $\mathbb{Z}_p^{(\ell-3)\times 2}$, i.e., $\mathsf{Rk}_2\left(\mathbb{Z}_p^{(\ell-3)\times 2}\right)$; $L : \mathbb{Z}_p^{\ell \times 2} \times \mathbb{Z}_p^{\ell \times 2} \to \{0, 1\}^{2\mu}$; recall that $2\mu = (\ell - 6) \cdot \log p - \omega(\kappa)$, and thus $|L| \leq p^{\ell-6} \cdot \kappa^{-\omega(1)}$. By Lemma 6, for any (even computationally unbounded) L, we have

$$\Delta\left(\left(A, X, L(A, X \cdot T, X \cdot T'), X \cdot T'\right), \left(A, X, L(A, U, X \cdot T'), X \cdot T'\right)\right) < \kappa^{-\omega(1)} = \mathsf{negl}(\kappa).$$

Let g be a random generator of G, and ω is some randomness chosen uniformly. We define a particular function L^*, with g, ω hardwired, as follows: $L^*(A, w, v)$ on input A, w, v does the following:

- It first samples $Y_0, \ldots, Y_{i-1} \leftarrow \mathsf{Ker}^2(A)$, using the random coins ω. Then it sets $\mathsf{sk}_j = g^{Y_j}$ for $j \in [i-1]$.
- It simulates the leakage functions, adaptively, obtains the values $f_1(\mathsf{sk}_0, \mathsf{sk}_1), \ldots, f_{i-1}(\mathsf{sk}_{i-2}, \mathsf{sk}_{i-1})$, and obtains the next leakage function f_i.
- It computes $f_i(\mathsf{sk}_{i-1}, g^w)$, and then obtains the next leakage function f_{i+1}.
- Finally it outputs $f_i(\mathsf{sk}_{i-1}, g^w)||f_{i+1}(g^w, g^v)$.

Recall that f_i, f_{i+1} are two leakage functions with μ bits of output, so L^* has 2μ bits of output. Now we construct the adversary $\mathcal{B}$ as follows:

- Let g be the random generator, ω be the random coins as stated above, and L^* be the function defined above. Then $\mathcal{B}$ gets input $(A, X, L^*(A, Z, X \cdot T'), X \cdot T')$ where Z is either uniformly random or $X \cdot T$.
- $\mathcal{B}$ samples $Y_0, \ldots, Y_{i-1} \leftarrow \mathsf{Ker}^2(A)$, using the random coins ω. Then it sets $\mathsf{sk}_j = g^{Y_j}$ for $j \in [i-1]$. We note that the secret keys (in the first $i-1$ rounds) are consistent with the values used in the leakage function for they use the same randomness ω.
- $\mathcal{B}$ sets $\mathsf{sk}_{i+2} = g^{X \cdot T'}$.
- $\mathcal{B}$ samples $T_{i+3}, \ldots, T_{t+1} \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{(\ell-3)\times 2})$ and sets $\mathsf{sk}_j = g^{X \cdot T_j}$ for $j \in \{i+3, \ldots, t+1\}$.
- $\mathcal{B}$ outputs $\mathcal{A}\Big(g^A, g^z, f_1(\mathsf{sk}_0, \mathsf{sk}_1), f_2(\mathsf{sk}_1, \mathsf{sk}_2), \cdots, f_{i-1}(\mathsf{sk}_{i-2}, \mathsf{sk}_{i-1}), L^*(Z, X \cdot T'), f_{i+2}(\mathsf{sk}_{i+2}, \mathsf{sk}'_{i+3}), \ldots, f_t(\mathsf{sk}'_t, \mathsf{sk}'_{t+1})\Big)$.

Then it is not hard to see that if Z comes from the distribution XT, then the simulation of $\mathcal{B}$ and L^* distributes as Q_i, and otherwise Q_{i-1}. Thus, suppose $\mathcal{A}$ can distinguish Q_i from Q_{i+1} with non-negligible probability ϵ, then $\mathcal{B}$ can distinguish the two distributions with a non-negligible probability. This contradicts Lemma 6.

Finally, we show that D''_0 is computationally indistinguishable from D''_1 under the linear assumption.

Lemma 13. *Under the linear assumption, the distributions D''_0 and D''_1 are computationally indistinguishable.*

We use the same argument as the work [12]. In particular, we will prove that suppose there exists an adversary $\mathcal{A}$ that distinguishes D''_0 from D''_1, then there exists an adversary $\mathcal{B}$ that distinguishes the distributions $\{g^C : C \leftarrow \mathbb{Z}_p^{3\times 3}\}$ and $\{g^C : C \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{3\times 3})\}$. We assume that the second distribution samples two random rows, and then sets the third row as a random linear combination of the first two rows. As argued in the work [12], this assumption is without loss of generality.

Now we describe the adversary $\mathcal{B}$. $\mathcal{B}$ on input g^C does the following.

- $\mathcal{B}$ samples a random matrix $X \leftarrow \mathbb{Z}_p^{\ell\times(\ell-3)}$, and a random matrix $B \leftarrow \mathbb{Z}_p^{3\times\ell}$ such that $B \cdot X = 0$.
- $\mathcal{B}$ computes g^{CB}, and sets its first two rows as g^A and the last row as g^z.
- $\mathcal{B}$ samples $T_1, \ldots, T_t \leftarrow \mathsf{Rk}_2(\mathbb{Z}_p^{(\ell-3)\times 2})$, and sets $\mathsf{sk}_i = g^{XT_i}$ for $i \in [t]$.

– $\mathcal{B}$ outputs $\mathcal{A}(g, g^A, g^z, f_1(\mathsf{sk}_0, \mathsf{sk}_1), \ldots, f_t(\mathsf{sk}_{t-1}, \mathsf{sk}_t))$.

As argued in the work [12], if C is uniformly random, then (A, z) is distributed uniformly as D_1''. If C is of rank 2, then (A, z) is distributed as $(A, r^T A)$ for some random $r \in \mathbb{Z}_p^2$ as D_0''. Thus, suppose $\mathcal{A}$ can distinguish D_0'' from D_1'' with non-negligible probability, then $\mathcal{B}$ breaks the linear assumption with non-negligible probability.

Lemma 9 ($D_0 \approx D_1$) follows directly from Lemmas 10, 12, and 13. This suffices to prove the theorem. We present the proofs of Lemmas 10, 12, and 13.

5. Leakage-Resilient PKE from Obfuscation

5.1. *Making Sahai–Waters PKE Leakage-Resilient*

We show that by modifying the Sahai–Waters (SW) public key encryption scheme [47] in two simple ways, the scheme already becomes non-trivially leakage resilient in the one-time, bounded setting. Recall that in this setting, the adversary, after seeing the public key and before seeing the challenge ciphertext, may request a single leakage query of length L bits. We require that semantic security hold, even given this leakage.

Our scheme can tolerate an arbitrary amount of one-time leakage. Specifically, for any $L = L(\kappa) = \mathrm{poly}(\kappa)$, we can obtain a scheme which is L-leakage resilient by setting the parameter ρ in Fig. 6 depending on L. However, our leakage *rate* is far from optimal, since the size of the secret key, sk, grows with L. Indeed, the result of this section is subsumed by the work of Hazay et al. [35]. We view this section as a warm-up; in Sect. 5.2, we will show how to further modify the construction to achieve optimal leakage rate, though, we rely upon much stronger assumptions than those of Hazay et al. [35].

At a high-level, we modify SW in the following ways: (1) Instead of following the general paradigm of encrypting a message m by xoring with the output of a PRF, we first apply a strong randomness extractor Ext to the output of the PRF and then xor with the message m; (2) we modify the secret key of the new scheme to be an iO of the underlying decryption circuit. Recall that in SW, decryption essentially consists of evaluating a puncturable PRF. In our scheme, sk consists of an iO of the puncturable PRF, padded with $\mathrm{poly}(L)$ bits.

We show that, even given L bits of leakage, the attacker cannot distinguish $\mathsf{Ext}(y)$ from random, where y is the output of the PRF on a fixed input t^*. This will be sufficient to prove security. We proceed by a sequence of hybrids: First, we switch sk to be an obfuscation of a circuit which has a PRF key punctured at t^* and a point function $t^* \to y$ hardcoded. On input $t \neq t^*$, the punctured PRF is used to compute the output, whereas on input t^*, the point function is used. Since the circuits compute the same function and—due to appropriate padding—they are both the same size, security of the iO implies that an adversary cannot distinguish the two scenarios. Next, just as in SW, we switch from $t^* \to y$ to $t^* \to y^*$, where y^* is uniformly random of length $L + L_{\mathsf{msg}} + 2\log(1/\epsilon)$ bits; here, we rely on the security of the punctured PRF. Now, observe that since y^* is uniform and since Ext is a strong extractor for inputs of min-entropy $L_{\mathsf{msg}} + 2\log(1/\epsilon)$ and output length L_{msg}, $\mathsf{Ext}(y^*)$ looks random, even under L bits of leakage (Figs. 7, 8).

Encryption scheme $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ using obfuscator iO and PRG G.

Key Generation: $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
Compute $k \leftarrow \mathsf{PRF}.\mathsf{Gen}(1^\kappa)$, where $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\rho \to \{0,1\}^\rho$. Let C_k be the circuit described in Figure 7, and let $C_{\mathsf{Enc}} \leftarrow \mathsf{iO}(C_k)$.
Let $C_{k,\kappa+\rho}$ be the circuit described in Figure 8, and let $C_{\mathsf{Dec}} \leftarrow \mathsf{iO}(C_{k,\kappa+\rho})$.
Output $\mathsf{pk} = (C_{\mathsf{Enc}})$ and $\mathsf{sk} = (C_{\mathsf{Dec}})$.

Encryption: $c \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m)$
On input message $m \in \{0,1\}^{L_{\mathsf{msg}}}$, sample $r \leftarrow \{0,1\}^\kappa, w \leftarrow \{0,1\}^d$, and output $c = (G(r), w, \mathsf{Ext}(C_{\mathsf{Enc}}(r), w) \oplus m)$, where PRG $G : \{0,1\}^\kappa \to \{0,1\}^\rho$, and $\mathsf{Ext} : \{0,1\}^\rho \times \{0,1\}^d \to \{0,1\}^{L_{\mathsf{msg}}}$.

Decryption: $\hat{m} \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c)$
On input ciphertext $c = (t, w, v)$, compute $y := C_{\mathsf{Dec}}(t)$.
If $y \neq \perp$, output $\hat{m} = \mathsf{Ext}(y, w) \oplus v$. Otherwise, output $\hat{m} = \perp$.

Fig. 6. One-time, bounded leakage encryption scheme, $\mathcal{E}$.

Internal (hardcoded) state: k.

On input: r

– Output $z = \mathsf{PRF}.\mathsf{Eval}(k, G(r))$, where G is the same PRG used in $\mathcal{E}.\mathsf{Enc}$.

Fig. 7. This program C_k is obfuscated using iO and placed in the public key to be used for encryption.

Internal (hardcoded) state: k.

On input: t

– Output $z = \mathsf{PRF}.\mathsf{Eval}(k, t)$.

Fig. 8. The circuit above is padded with poly$(\kappa + \rho)$ dummy gates to obtain the circuit $C_{k,\kappa+\rho}$. $C_{k,\kappa+\rho}$ is then obfuscated using iO and placed in the secret key.

Theorem 9. *Assume*

– PRG $G : \{0,1\}^\kappa \to \{0,1\}^\rho$ *is a pseudorandom generator with output length* $\rho \geq 2\kappa$.
– $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\rho \to \{0,1\}^\rho$ *is a puncturable pseudorandom function.*
– iO *is indistinguishability obfuscator for circuits in this scheme.*
– $\mathsf{Ext} : \{0,1\}^\rho \times \{0,1\}^d \to \{0,1\}^{L_{\mathsf{msg}}}$ *is a* $(L_{\mathsf{msg}} + 2\log(1/\epsilon), \epsilon)$*-strong extractor, where* $\epsilon = \mathsf{negl}(\kappa)$.

Then $\mathcal{E}$ is L-leakage resilient against one-time key leakage where

$$L = \rho - 2\log(1/\epsilon) - L_{\mathsf{msg}}$$

Note that in the above theorem statement, ρ can be increased arbitrarily while all other parameters remain fixed. Therefore, to achieve an arbitrary amount, L, of leakage,

we fix L_{msg} and ϵ and then set $\rho := L + 2\log(1/\epsilon) + L_{\mathsf{msg}}$. Additionally, note that extractors that satisfy the requirements of Theorem 9 can be constructed via the Leftover Hash Lemma (c.f. [36]).

In order to prove Theorem 9, we prove (in Lemma 14) that even under leakage, it is hard for any PPT adversary $\mathcal{A}$ to distinguish the output of the extractor, Ext from uniform random. Given this, Theorem 9 follows immediately.

Lemma 14. *For every* PPT *leaking adversary $\mathcal{A}$, who is given oracle access to a leakage oracle $\mathcal{O}$ and may leak at most $\rho - 2\log(1/\epsilon) - L_{\mathsf{msg}}$ bits of the secret key, there exist random variables $\mathsf{pk}', \widetilde{\mathsf{sk}}$ such that:*

$$\left(\mathsf{pk}, t, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk})\right)$$
$$\overset{c}{\approx} \left(\mathsf{pk}', U_\rho, w, U_{L_{\mathsf{msg}}}, f(\widetilde{\mathsf{sk}}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$$

where $y = C_{\mathsf{Dec}}\mathsf{ct}_{\mathsf{dummy}}$, $(t = G(r))$ and the distributions are taken over coins of $\mathcal{A}$ and choice of $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, w, r and choice of $\mathsf{pk}', \widetilde{\mathsf{sk}}$, w, respectively.

We prove the lemma via the following sequence of hybrids: Note that Hybrids 1, 2 are essentially identical to the Sahai–Waters hybrids. We differ from Sahai–Waters when we modify the secret key in Hybrids 3 and 4.

Hybrid 0: This hybrid is identical to the real game.

Let $\mathcal{D}^{\mathcal{A}}_{H_0}$ denote the distribution $(\mathsf{pk}, t, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}))$ as in the left side of Lemma 14.

Hybrid 1: This hybrid is the same as Hybrid 0 except we replace pseudorandom $t = G(r)$ in the challenge ciphertext with uniform random $t^* \leftarrow \{0,1\}^\rho$.

Let $\mathcal{D}^{\mathcal{A}}_{H_1}$ denote the distribution $(\mathsf{pk}, t^*, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}))$ where $y = C_{\mathsf{Dec}}(t^*)$ and the distribution is taken over coins of $\mathcal{A}$, choice of $(\mathsf{pk}, \mathsf{sk})$, w, t^* as described above (Fig. 9).

Claim 1. *For every* PPT *adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_0} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_1}.$$

Proof. The proof is by reduction to the security of the pseudorandom generator G. Assume toward contradiction that there exists a PPT adversary $\mathcal{A}$, a corresponding PPT distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many κ, D distinguishes $\mathcal{D}^{\mathcal{A}}_{H_0}$ and $\mathcal{D}^{\mathcal{A}}_{H_1}$ with probability at least $1/p(\kappa)$. We construct a PPT adversary $\mathcal{S}$ that distinguishes the output of the PRG from uniform random with probability at least $1/p(\kappa)$, for infinitely many κ. $\mathcal{S}$ does the following: $\mathcal{S}$ runs $\mathcal{E}.\mathsf{Gen}(1^\kappa)$ honestly to generate $(\mathsf{pk}, \mathsf{sk})$. $\mathcal{S}$ hands pk to $\mathcal{A}$ and responds to leakage query f by applying the leakage function directly to sk to compute $f(\mathsf{sk})$. Upon receiving its challenge t' as the external PRG challenge, $\mathcal{S}$ sets $y = C_{\mathsf{Dec}}(t')$, hands $(\mathsf{pk}, t', w, \mathsf{Ext}(y, w), f(\mathsf{sk}))$

Internal (hardcoded) state: $\widetilde{k} = \mathsf{PRF.Punct}(k, t^*)$.

On input: r

– Output $z = \mathsf{PRF.Eval}(k, G(r))$, where G is the same PRG used in $\mathcal{E}.\mathsf{Enc}$.

Fig. 9. Program $C_{\widetilde{k}}$. This program replaces C_k. It is obfuscated and placed in the public key to be used for encryption.

to the distinguisher D, and outputs whatever D does. The reader can verify that $\mathcal{S}$'s distinguishing advantage is the same as D's. □

Hybrid 2: This hybrid is the same as Hybrid 1 except we replace the key k used in C_{Enc} with a punctured key, $\widetilde{k} = \mathsf{PRF.Punct}(k, t^*)$, and denote it as C'_{Enc}.

We denote the resulting public key by pk'.

Let $\mathcal{D}^{\mathcal{A}}_{H_2}$ denote the distribution $\left(\mathsf{pk}', t^*, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$ where $y = C_{\mathsf{Dec}}(t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \mathsf{sk})$, w, t^* as described above.

Claim 2. *For every* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_1} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_2}.$$

Proof. The proof is by a reduction to the security of the indistinguishability obfuscation. The main observation is that with all but negligible probability, t^* is not in the range of the PRG , in which case C_{Enc} and the modified circuit C'_{Enc} used in Hybrid 2 have identical behavior. Thus, with high probability for all inputs neither program can call on $\mathsf{PRF.Punct}(k, t^*)$. Therefore, puncturing t^* out from the key k will not effect the input/output behavior. Therefore, if there is a difference in advantage, we can create an algorithm $\mathcal{B}$ that breaks the security of indistinguishability obfuscation.

$\mathcal{B}$ runs as the challenger, but where t^* is chosen at random. When it is to create the obfuscated program, it submits both programs $C_0 = C_k$ and $C_1 = C_{\widetilde{k}}$ to an iO challenger. If the iO challenger chooses the first, then we are in Hybrid 1. If it chooses the second, then we are in Hybrid 2. Any adversary with non-negligible advantages in the two hybrids leads to $\mathcal{B}$ as an attacker on iO security. □

Hybrid 3: This hybrid is the same as Hybrid 2 except we replace $C_{\mathsf{Dec}} = \mathsf{iO}(C_{k,\kappa+\rho})$ with $C'_{\mathsf{Dec}} = \mathsf{iO}(C'_{\widetilde{k}})$, where $C'_{\widetilde{k}}$ is specified in Fig. 10. Note that we puncture k at the challenge point t^*. We denote by sk' the resulting secret key.

Let $\mathcal{D}^{\mathcal{A}}_{H_3}$ denote the distribution $\left(\mathsf{pk}', t^*, w, \mathsf{Ext}(y, w), f(\mathsf{sk}') \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widehat{\mathsf{pk}})\right)$ where $y = C'_{\mathsf{Dec}}(t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}, \mathsf{sk}')$, w, t^* as described above.

Claim 3. *For every* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_2} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_3}.$$

Internal (hardcoded) state: $(t^*, \beta = \mathsf{PRF.Eval}(k, t^*)), \widetilde{k} = \mathsf{PRF.Punct}(k, t^*)$.

On input: t

- If $t = t^*$, output β.
- Otherwise, output $\mathsf{PRF.Eval}(\widetilde{k}, t)$.

Fig. 10. Program $C'_{\widetilde{k}}$. This program replaces $C_{k,\kappa+\rho}$. It is obfuscated and placed in the secret key.

Proof. The proof is by a reduction to the security of the indistinguishability obfuscation. The main observation is that the size of the circuit does not change since the description of $C_{k,\kappa+\rho}$ is padded with $\mathrm{poly}(\kappa + \rho)$ gates (for appropriate poly). Thus, $C_{k,\kappa+\rho}$ and $C'_{\widetilde{k}}$ are the same size. Moreover, puncturing t^* out from the key k will not effect the input/output behavior since on input t^* we output the hardcoded value $\beta = \mathsf{PRF.Eval}(k, t^*)$. Therefore, if there is a difference in advantage, we can create an algorithm $\mathcal{B}$ that breaks the security of indistinguishability obfuscation.

$\mathcal{B}$ runs as the challenger, but where t^* is chosen at random. When it is to create the obfuscated program it submits both programs $C_0 = C_{k,\kappa+\rho}$ and $C_1 = C'_{\widetilde{k}}$ to an iO challenger. If the iO challenger chooses the first then we are in Hybrid 1. If it chooses the second then we are in Hybrid 2. Any adversary with non-negligible advantages in the two hybrids leads to $\mathcal{B}$ as an attacker on iO security. □

Hybrid 4: This hybrid is the same as Hybrid 3 except we replace the hardcoded β with y^*, where y^* is uniformly random. We denote by $\widetilde{\mathsf{sk}}$ the resulting secret key. Note that the public key pk' remains the same.

Let $\mathcal{D}^{\mathcal{A}}_{H_4}$ denote the distribution $\left(\mathsf{pk}', t^*, w, \mathsf{Ext}(y^*, w), f(\widetilde{\mathsf{sk}}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$ where $y^* = C'_{\mathsf{Dec}}(t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \widetilde{\mathsf{sk}}), w, t^*$ as described above.

Claim 4. *For every* PPT *adversary* $\mathcal{A}$,

$$\mathcal{D}^{\mathcal{A}}_{H_3} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_4}.$$

Proof. The proof is through a reduction to the security of the puncturable PRF. Recall, the security notion of puncturable PRFs states that, given $\mathsf{PRF.Punct}(k, t^*)$, an adversary cannot distinguish $\mathsf{PRF.Eval}(k, t^*)$ from random. The reduction is straightforward: to break the security of the PRF, $\mathcal{S}$ generates t^* at random and submits it to his challenger. He receives $\mathsf{PRF.Punct}(k, t^*)$, along with either $y^* = \mathsf{PRF.Eval}(k, t^*)$ or $y^* \leftarrow \{0,1\}^{\rho}$ as a challenge. He uses y^*, and samples all the remaining necessary keys for simulating $\widehat{\mathsf{pk}}$ and $\widehat{\mathsf{sk}}$. He chooses w at random and computes $\mathsf{Ext}(y^*, w)$. He answers leakage queries on $\widetilde{\mathsf{sk}}$ honestly. The reader can verify that $\mathcal{S}$'s advantage is the same as $\mathcal{A}$'s advantage in distinguishing the two hybrids. □

Claim 5.

$$\mathcal{D}^{\mathcal{A}}_{H_4} \overset{s}{\approx} \left(\mathsf{pk}', U_{\rho}, w, U_{L_{\mathsf{msg}}}, f(\widetilde{\mathsf{sk}}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widetilde{\mathsf{pk}})\right)$$

Note that the right side above is the same as the right side of Lemma 14

Proof. We claim that the min-entropy of y^* conditioned on $\mathsf{pk}', f(\widetilde{\mathsf{sk}})$ is at least $L_{\mathsf{msg}} + 2\log(1/\epsilon)$. Note that y^* initially has min-entropy ρ since it is chosen uniformly at random. Thus, leaking $\rho - 2\log(1/\epsilon) - L_{\mathsf{msg}}$ number of bits of the secret key reduces y^*'s min-entropy by at most $\rho - 2\log(1/\epsilon) - L_{\mathsf{msg}}$. Therefore, y^* maintains min-entropy at least $L_{\mathsf{msg}} + 2\log(1/\epsilon)$. and so the claim follows by the properties of the strong extractor, Ext. □

This concludes the proof of Lemma 14.

5.2. *Improving the Leakage Rate*

In this section, we show how to modify the previous construction to achieve optimal leakage rate. The key observation is that the leakage rate tolerated by the previous construction is low because the entire obfuscated circuit $\mathsf{iO}(C_{k,\kappa+\rho})$ must be stored in the secret key. Ideally, since the circuit is obfuscated, we would like to put it in the public key. However, this cannot possibly work since anyone can then decrypt the challenge ciphertext. Therefore, we store a collision-resistant hash $h(\mathsf{ct}_{\mathsf{dummy}})$ in the obfuscated circuit, and include a ciphertext encrypted using a symmetric key encryption scheme, $\mathsf{ct}_{\mathsf{dummy}}$, in the secret key: the circuit will only decrypt if the user provides a proper pre-image to the hardcoded value $h(\mathsf{ct}_{\mathsf{dummy}})$. This scheme seems to preserve semantic security, but we must prove security in the LR setting. Specifically, we must show that even when leaking $1 - o(1)$-fraction of $\mathsf{ct}_{\mathsf{dummy}}$, the adversary cannot find a valid input to the obfuscated circuit. To prove this, the idea is that in the hybrids, we switch $\mathsf{ct}_{\mathsf{dummy}}$ from a "dummy input" to an encryption of the point function $t^* \to y^*$, where y^* is random. The obfuscated circuit will also be changed (as in the proof of the previous construction) so that on input t^*, it outputs the output of the point function. Note that even under leakage, y^* has high min-entropy and thus $\mathsf{Ext}(y^*)$ will still look random. Finally, we note that in order for the argument to work, we must now rely on public-coin differing-inputs obfuscation, since in the hybrid arguments the obfuscated circuits in the public key will produce different outputs on inputs $\mathsf{ct}_{\mathsf{dummy}} \neq \mathsf{ct}'_{\mathsf{dummy}}$, such that $h(\mathsf{ct}'_{\mathsf{dummy}}) = h(\mathsf{ct}_{\mathsf{dummy}})$, which are hard for an efficient adversary to find (Figs. 11, 12, 13, 14) .

Theorem 10. *Assume*

- E *is a semantically secure symmetric key encryption scheme with ciphertexts of length* $L_{\mathsf{ct}}(\kappa, L_{\mathsf{msg}})$ *for* L_{msg}*-bit messages and security parameter* κ.
- h *is a collision-resistant hash function. with output length* $L_{\mathsf{h}}(\kappa)$ *for security parameter* κ.
- PRG $G : \{0,1\}^\kappa \to \{0,1\}^\rho$ *is a pseudorandom generator with output length* $\rho \geq 2\kappa$.
- $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\rho \to \{0,1\}^\rho$ *is a puncturable pseudorandom function.*
- diO *is a public-coin, differing-inputs obfuscator for circuits in this scheme.*
- $\mathsf{Ext} : \{0,1\}^\rho \times \{0,1\}^d \to \{0,1\}^{L_{\mathsf{msg}}}$ *is a* $(L_{\mathsf{msg}} + 2\log(1/\epsilon), \epsilon)$*-strong extractor, where* $\epsilon = \mathsf{negl}(\kappa)$.

Encryption Scheme $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$

Key Generation: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$

Compute the following:

- $(\mathsf{sk}_\mathsf{E}) \leftarrow \mathsf{E.Gen}(1^\kappa)$,
- $h \leftarrow \mathcal{H}$,
- $k \leftarrow \mathsf{PRF.Gen}(1^\kappa)$, where $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\rho \rightarrow \{0,1\}^\rho$.
- $\mathsf{ct}_\mathsf{dummy} \leftarrow \mathsf{E.Enc}(\mathsf{sk}_\mathsf{E}, 0^\kappa || 0^\rho; r_0)$, and $h^* = h(\mathsf{ct}_\mathsf{dummy})$.

Let C_k be the circuit described in Figure 12, and let $C_\mathsf{Enc} \leftarrow \mathsf{diO}(C_k)$.
Let $\mathsf{keys} = \{\mathsf{sk}_\mathsf{E}, h^*\}$, let C_keys be the circuit described in Figure 13, and let $C_\mathsf{Dec} \leftarrow \mathsf{diO}(C_\mathsf{keys})$.
Output $\mathsf{pk} = (C_\mathsf{Enc}, C_\mathsf{Dec})$ and $\mathsf{sk} = (\mathsf{ct}_\mathsf{dummy})$.

Encryption: $c \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m)$

On input message $m \in \{0,1\}^{L_\mathsf{msg}}$, sample $r \leftarrow \{0,1\}^\kappa, w \leftarrow \{0,1\}^d$, and output $c = (G(r), w, \mathsf{Ext}(C_\mathsf{Enc}(r), w) \oplus m)$, where PRG $G : \{0,1\}^\kappa \rightarrow \{0,1\}^\rho$, and $\mathsf{Ext} : \{0,1\}^\rho \times \{0,1\}^d \rightarrow \{0,1\}^{L_\mathsf{msg}}$.

Decryption: $\hat{m} \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c)$

On input ciphertext $c = (t, w, v)$, compute $y := C_\mathsf{Dec}(\mathsf{ct}_\mathsf{dummy}, t)$.
If $y \neq \perp$, output $\hat{m} = \mathsf{Ext}(y, w) \oplus v$. Otherwise, output $\hat{m} = \perp$.

Fig. 11. One-time, bounded leakage encryption scheme,$\mathcal{E}$.

Internal (hardcoded) state: k.

On input: r

- Output $z = \mathsf{PRF.Eval}(k, G(r))$, where G is the same PRG used in $\mathcal{E}.\mathsf{Enc}$.

Fig. 12. This program C_k is obfuscated and placed in the public key to be used for encryption.

Internal (hardcoded) state: $\mathsf{keys} = \{k, h^*\}$.

On input: $\mathsf{ct}_\mathsf{dummy}, t$

- If $h(\mathsf{ct}_\mathsf{dummy}) \neq h^*$ output $\perp$.
- Otherwise, output $z = \mathsf{PRF.Eval}(k, t)$.

Fig. 13. This program C_keys is obfuscated and placed in the public key. It is used during decryption.

Internal (hardcoded) state: $\widetilde{k} = \mathsf{PRF.Punct}(k, t^*)$.

On input: r

- Output $z = \mathsf{PRF.Eval}(k, G(r))$, where G is the same PRG used in $\mathcal{E}.\mathsf{Enc}$.

Fig. 14. Program $C_{\widetilde{k}}$. This program replaces C_k. It is obfuscated and placed in the public key to be used for encryption.

Then $\mathcal{E}$ is L-leakage resilient against one-time key leakage where

$$L = |\mathsf{sk}| \cdot \frac{\rho - 2\log(1/\epsilon) - L_{\mathsf{msg}} - L_{\mathsf{h}}(\kappa)}{(L_{\mathsf{ct}}(\kappa, \kappa + \rho))}$$

Proof. First, note that extractors that satisfy the requirements of Theorem 10 can be constructed via the Leftover Hash Lemma (c.f. [36]). We can choose a semantically secure symmetric key encryption scheme with $L_{\mathsf{ct}}(\kappa, \kappa + \rho) = O(\kappa) + \kappa + \rho$, for messages of length $\kappa + \rho$, as this is achieved by appropriate modes of operation. Finally, choosing a collision-resistant hash function h with output length $L_{\mathsf{h}}(\kappa) = O(\kappa)$, and setting $\rho = \omega(\kappa)$, $\epsilon = 2^{-\Theta(\kappa)}$, $L_{\mathsf{msg}} = \Theta(\kappa)$, yields an encryption scheme for messages of length $\Theta(\kappa)$ with leakage rate $1 - o(1)$.

In order to prove Theorem 10, we prove (in Lemma 15) that even under leakage, it is hard for any PPT adversary $\mathcal{A}$ to distinguish the output of the extractor, Ext from uniform random. Given this, Theorem 10 follows immediately. □

Lemma 15. *For every* PPT *leaking adversary $\mathcal{A}$, who is given oracle access to a leakage oracle $\mathcal{O}$ and may leak at most $\rho - 2\log(1/\epsilon) - L_{\mathsf{msg}}$ bits of the secret key, there exist random variables $\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}$ such that:*

$$\left(\mathsf{pk}, t, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk})\right)$$
$$\overset{c}{\approx} \left(\widetilde{\mathsf{pk}}, U_\rho, w, U_{L_{\mathsf{msg}}}, f(\widetilde{\mathsf{sk}}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widetilde{\mathsf{pk}})\right)$$

where $y = C_{\mathsf{Dec}}(\mathsf{ct}_{\mathsf{dummy}}, t = G(r))$ and the distributions are taken over coins of $\mathcal{A}$ and choice of $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, w, r and choice of $\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}$, w, respectively.

We prove the lemma via the following sequence of hybrids:

Hybrid 0: This hybrid is identical to the real game.

Let $\mathcal{D}^{\mathcal{A}}_{H_0}$ denote the distribution $(\mathsf{pk}, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}))$ as in the left side of Lemma 15.

Hybrid 1: This hybrid is the same as Hybrid 0 except we replace pseudorandom $t = G(r)$ in the challenge ciphertext with uniform random $t^* \leftarrow \{0,1\}^\rho$.

Let $\mathcal{D}^{\mathcal{A}}_{H_1}$ denote the distribution $(\mathsf{pk}, t^*, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}))$ where $y = C_{\mathsf{Dec}}(\mathsf{ct}_{\mathsf{dummy}}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, choice of $(\mathsf{pk}, \mathsf{sk})$, w, t^* as described above.

Claim 6. *For every* PPT *adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_0} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_1}.$$

Proof. The proof is by reduction to the security of the pseudorandom generator G. Assume toward contradiction that there exists a PPT adversary $\mathcal{A}$, a corresponding PPT distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many κ, D distinguishes $\mathcal{D}^{\mathcal{A}}_{H_0}$ and $\mathcal{D}^{\mathcal{A}}_{H_1}$ with probability at least $1/p(\kappa)$. We construct a PPT adversary $\mathcal{S}$ that

distinguishes the output of the PRG from uniform random with probability at least $1/p(\kappa)$, for infinitely many κ. $\mathcal{S}$ does the following: $\mathcal{S}$ runs $\mathcal{E}.\mathsf{Gen}(1^\kappa)$ honestly to generate $(\mathsf{pk}, \mathsf{sk})$. $\mathcal{S}$ hands pk to $\mathcal{A}$ and responds to leakage query f by apply the leakage function directly to sk to compute $f(\mathsf{sk})$. Upon receiving its challenge t' as the external PRG challenge, $\mathcal{S}$ sets $y = C_{\mathsf{Dec}}(\mathsf{ct}_{\mathsf{dummy}}, t')$, hands $(\mathsf{pk}, t', w, \mathsf{Ext}(y, w), f(\mathsf{sk}))$ to the distinguisher D, and outputs whatever D does. The reader can verify that $\mathcal{S}$'s distinguishing advantage is the same as D's. □

Hybrid 2: This hybrid is the same as Hybrid 1 except we replace the key k used in C_{Enc} with a punctured key, $\widetilde{k} = \mathsf{PRF.Punct}(k, t^*)$, and denote it as C'_{Enc}. We denote the resulting public key by pk'.

Let $\mathcal{D}^{\mathcal{A}}_{H_2}$ denote the distribution $\left(\mathsf{pk}', t^*, w, \mathsf{Ext}(y, w), f(\mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$ where $y = C_{\mathsf{Dec}}(\mathsf{ct}_{\mathsf{dummy}}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \mathsf{sk})$, w, t^* as described above.

Claim 7. *For every* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_1} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_2}.$$

Proof. The proof is by a reduction to the security of the indistinguishability obfuscation (iO). The main observation is that with all but negligible probability, t^* is not in the range of the PRG, in which case C_{Enc} and the modified circuit C'_{Enc} used in Hybrid 2 have identical behavior. Thus, with high probability for all inputs neither program can call on $\mathsf{PRF.Punct}(k, t^*)$. Therefore, puncturing t^* out from the key k will not effect the input/output behavior. Therefore, if there is a difference in advantage, we can create an algorithm $\mathcal{B}$ that breaks the security of indistinguishability obfuscation.

$\mathcal{B}$ runs as the challenger, but where t^* is chosen at random. When it is to create the obfuscated program it submits both programs $C_0 = C_k$ and $C_1 = C_{\widetilde{k}}$ to an iO challenger. If the iO challenger chooses the first, then we are in Hybrid 1. If it chooses the second, then we are in Hybrid 2. Any adversary with non-negligible advantages in the two hybrids leads to $\mathcal{B}$ as an attacker on iO security. Note that since we only require indistinguishability obfuscation (iO) for this hybrid, it is actually sufficient in our construction to replace C_{Enc} with $C_{\mathsf{Enc}} \leftarrow \mathsf{iO}(C_k)$, i.e., we require only iO obfuscation, rather than diO obfuscation for this program. □

Hybrid 3: This hybrid is the same as Hybrid 2 except:

- we replace $\mathsf{ct}_{\mathsf{dummy}}$ with $\mathsf{ct}''_{\mathsf{dummy}}$, where $\mathsf{ct}''_{\mathsf{dummy}}$ is an encryption of $t^*||y$ and $y = \mathsf{PRF.Eval}(k, t^*)$.
- we replace h^* with $h''^* = h(\mathsf{ct}''_{\mathsf{dummy}})$.

We denote the resulting public key by pk'' and the resulting secret key by sk''.

Let $\mathcal{D}^{\mathcal{A}}_{H_3}$ denote the distribution $\left(\mathsf{pk}'', t^*, w, \mathsf{Ext}(y, w), f(\mathsf{sk}'') \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}'')\right)$ where $y = C_{\mathsf{Dec}}(\mathsf{ct}''_{\mathsf{dummy}}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}'', \mathsf{sk}'')$, w, t^* as described above.

Internal (hardcoded) state: $\mathsf{keys}' = \{\mathsf{sk}_{\mathsf{E}}, \widetilde{k} = \mathsf{PRF.Punct}(k, t^*), h''^*\}$.

On input: $\mathsf{ct}_{\mathsf{dummy}}, t$

- If $h(\mathsf{ct}_{\mathsf{dummy}}) \neq h''^*$ output $\perp$.
- Compute $\alpha || \beta = \mathsf{E.Dec}(\mathsf{sk}_{\mathsf{E}}, \mathsf{ct}_{\mathsf{dummy}})$.
- If $t = \alpha$, output β.
- Otherwise, output $\mathsf{PRF.Eval}(\widetilde{k}, t)$.

Fig. 15. Program $C_{\mathsf{keys}'}$. This program replaces C_{keys}. It is obfuscated and placed in the public key. It is used during decryption.

Claim 8. *For every* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_2} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_3}.$$

The proof is by a reduction to the semantic security of E.

Hybrid 4: This hybrid is the same as Hybrid 3 except we replace $C_{\mathsf{Dec}} = \mathsf{diO}(C_{\mathsf{keys}})$ with $C'_{\mathsf{Dec}} = \mathsf{diO}(C_{\mathsf{keys}'})$, where $C_{\mathsf{keys}'}$ is specified in Fig. 15. We denote by $\widehat{\mathsf{pk}}$ the resulting public key.

Let $\mathcal{D}^{\mathcal{A}}_{H_4}$ denote the distribution $\left(\widehat{\mathsf{pk}}, t^*, w, \mathsf{Ext}(y, w), f(\mathsf{sk}'') \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widehat{\mathsf{pk}})\right)$ where $y = C'_{\mathsf{Dec}}(\mathsf{ct}''_{\mathsf{dummy}}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\widehat{\mathsf{pk}}, \mathsf{sk}'')$, w, t^* as described above.

Claim 9. *For every* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_3} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_4}.$$

Proof. We define the following sampler Samp and show that the circuit family $\mathcal{C}$ associated with Samp is a differing-inputs circuit family.

$\mathsf{Samp}(1^\kappa)$ does the following:

- Set $\mathsf{keys} = (k, h''^*)$ and set $\mathsf{keys}' = (\mathsf{sk}_{\mathsf{E}}, \widetilde{k}, h''^*)$.
- Let $C_0 = C_{\mathsf{keys}}$ and let $C_1 = C_{\mathsf{keys}'}$.
- Set $\mathsf{aux} = (\mathsf{sk}_{\mathsf{E}}, h, h''^*, \mathsf{ct}''_{\mathsf{dummy}}, r\ t^*, y)$, where r is the randomness used for $\mathsf{ct}''_{\mathsf{dummy}}$.
- Return (C_0, C_1, aux).

Note that aux contains all of the random coins used by Samp.

We now show that for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[C_0(x) \neq C_1(x) : (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Samp}(1^\kappa), \\ x \leftarrow \mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})] \leq \mathsf{negl}(\kappa).$$

Assume toward contradiction that there exists a PPT adversary $\mathcal{A}$ and a polynomial $p(\cdot)$ such that for infinitely many κ, $\mathcal{A}$ outputs a distinguishing input with probability at least $1/p(\kappa)$. We construct a PPT adversary $\mathcal{S}$ that finds a collision on h.

On input $h \leftarrow \mathcal{H}$, $\mathcal{S}$ does the following:

- $\mathcal{S}$ simulates Samp by doing the following:
  - Run $(\mathsf{sk}_{\mathsf{E}}) \leftarrow \mathsf{E.Gen}(1^\kappa)$, $k \leftarrow \mathsf{PRF.Gen}(1^\kappa)$. Choose t^* at random and set $\widetilde{k} = \mathsf{PRF.Punct}(k, t^*)$.
  - $\mathcal{S}$ computes $y = \mathsf{PRF.Eval}(k, t^*)$ to generate $\mathsf{ct}''_{\mathsf{dummy}}$. It computes $h^* = h(\mathsf{ct}''_{\mathsf{dummy}})$.
  - Set $\mathsf{keys} = (k, h''^*)$ and $\mathsf{keys}' = (\mathsf{sk}_{\mathsf{E}}, \widetilde{k}, h''^*)$.
  - Let $C_0 = C_{\mathsf{keys}}$ and let $C_1 = C_{\mathsf{keys}'}$.
  - Set $\mathsf{aux} = (\mathsf{sk}_{\mathsf{E}}, h, h''^*\, \mathsf{ct}''_{\mathsf{dummy}}, r\ t^*, y)$.
- $\mathcal{S}$ runs $\mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})$ and receives x in return.
- $\mathcal{S}$ parses x as $(m,\ t)$ and outputs (m).

Note that $C_0(\mathsf{ct}''_{\mathsf{dummy}}, \cdot)$ and $C_1(\mathsf{ct}''_{\mathsf{dummy}}, \cdot)$ are functionally equivalent. Furthermore, on any input (m, t) where $h(m) \neq h''^*$, both circuits output $\perp$. Therefore, if $\mathcal{A}$ finds a distinguishing input $x = (m, t)$, then it must be the case that both of the following hold:

- $(m \neq \mathsf{ct}''_{\mathsf{dummy}})$
- $h(m) = h''^*$.

Thus, whenever $\mathcal{A}$ outputs a differing-inputs, $\mathcal{S}$ successfully finds a collision on h. Therefore, we have that for infinitely many κ, $\mathcal{S}$ outputs a collision with probability at least $1/p(\kappa)$.

Claim 9 follows from the fact that diO is a public-coin differing-inputs obfuscator and from the fact that the circuit family $\mathcal{C}$ associated with Samp is a differing-inputs family. This is the case since $\mathcal{D}^{\mathcal{A}}_{H_3}$ can be simulated given $(\mathsf{diO}(C_0), \mathsf{aux})$ and $\mathcal{D}^{\mathcal{A}}_{H_4}$ can be simulated given $(\mathsf{diO}(C_1), \mathsf{aux})$. □

Hybrid 5: This hybrid is the same as Hybrid 4 except we replace $\mathsf{ct}''_{\mathsf{dummy}}$ with $\widetilde{\mathsf{ct}}_{\mathsf{dummy}}$, where $\widetilde{\mathsf{ct}}_{\mathsf{dummy}}$ is an encryption of $t^*||y^*$, where y^* is uniformly random. We denote by $\widetilde{\mathsf{sk}}$ the resulting secret key. We replace h^* with $\widetilde{h}^* = h(\widetilde{\mathsf{ct}}_{\mathsf{dummy}})$ and denote by $\widetilde{\mathsf{pk}}$ the updated public key.

Let $\mathcal{D}^{\mathcal{A}}_{H_5}$ denote the distribution $\left(\widetilde{\mathsf{pk}}, t^*, w, \mathsf{Ext}(y^*, w), f(\widetilde{\mathsf{sk}}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widetilde{\mathsf{pk}})\right)$ where $y^* = C'_{\mathsf{Dec}}(\widetilde{\mathsf{ct}}_{\mathsf{dummy}}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$, w, t^* as described above.

Claim 10. *For every* PPT *adversary* $\mathcal{A}$,

$$\mathcal{D}^{\mathcal{A}}_{H_4} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_5}.$$

Proof. The proof is through a reduction to the security of the puncturable PRF. Recall, the security notion of puncturable PRFs states that, given $\mathsf{PRF.Punct}(k, t^*)$, an adversary cannot distinguish $\mathsf{PRF.Eval}(k, t^*)$ from random. The reduction is straightforward: to break the security of the PRF, $\mathcal{S}$ generates t^* at random and submits it to his challenger. He receives $\mathsf{PRF.Punct}(k, t^*)$, along with either $y^* = \mathsf{PRF.Eval}(\kappa, t^*)$ or $y^* \leftarrow \{0,1\}^\rho$ as a challenge. He uses y^*, and samples all the remaining necessary

keys for simulating $\widehat{\mathsf{pk}}$ and $\widehat{\mathsf{sk}}$. He chooses w at random and computes $\mathsf{Ext}(y^*, w)$. He answers leakage queries on $\widehat{\mathsf{sk}}$ honestly. The reader can verify that $\mathcal{S}$'s advantage is the same as $\mathcal{A}$'s advantage in distinguishing the two hybrids.

Claim 11.

$$\mathcal{D}^{\mathcal{A}}_{H_5} \overset{s}{\approx} \left(\widetilde{\mathsf{pk}}, U_\rho, w, U_{L_{\mathsf{msg}}}, f(\widetilde{\mathsf{sk}}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widetilde{\mathsf{pk}})\right)$$

Note that the right side above is the same as the right side of Lemma 15

Proof. We claim that the min-entropy of y^* conditioned on $\widetilde{\mathsf{pk}}, f(\widetilde{\mathsf{sk}})$ is at least $L_{\mathsf{msg}} + 2\log(1/\epsilon)$. Note that y^* initially has min-entropy ρ since it is chosen uniformly at random. Recall that $\mathsf{ct}_{\mathsf{dummy}}$ has length $L_{\mathsf{ct}}(\kappa, \kappa + \rho)$, and h has output length $L_{\mathsf{h}}(\kappa)$. Thus, conditioning on $\widetilde{\mathsf{pk}}$ reduces y^*'s min-entropy by at most $L_{\mathsf{h}}(\kappa)$ (since only $\widetilde{h}^*$ contains information about y^*). Moreover, leaking another $\rho - 2\log(1/\epsilon) - L_{\mathsf{msg}} - L_{\mathsf{h}}(\kappa)$ number of bits of $\mathsf{ct}_{\mathsf{dummy}}$ reduces y^*'s min-entropy further by at most $\rho - 2\log(1/\epsilon) - L_{\mathsf{msg}}$. Therefore, y^* maintains min-entropy at least $L_{\mathsf{msg}} + 2\log(1/\epsilon)$, and the claim follows by the properties of the strong extractor, Ext. □

This concludes the proof of Lemma 15. □

6. Continual Leakage Resilience for One-Way Relations

Dodis et al. [21] defined one-way relation (OWR) in the regular continual leakage resilience setting, and present a construction based on a simpler primitive – leakage-indistinguishable re-randomizable relation (LIRR). In this section, we first extend their definition to *2CLR* and *CLR with leakage on key updates*. Then we prove their LIRR-based construction actually achieves the 2CLR security. By using our generic transformation from Sect. 3, we can have a construction for achieving CLR with leakage on key updates. Additionally, we give a new construction of 2CLR OWR based on 2CLR PKE, which can be obtained from the previous sections.

6.1. *Continual Leakage Model*

A one-way relation scheme OWR consists of two algorithms: $\mathsf{OWR.Gen}$ and $\mathsf{OWR.Verify}$. In the continual leakage setting, we require an additional algorithm $\mathsf{OWR.Update}$ which updates the secret keys. Note that the public key remains unchanged.

- $\mathsf{OWR.Gen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk}_0)$. The key generation algorithm takes in the security parameter κ, and outputs a secret key sk_0 and a public key pk.
- $\mathsf{OWR.Verify}(\mathsf{pk}, \mathsf{sk}) \to \{0, 1\}$. The verification algorithm takes in the public key pk, a secret key sk, and outputs either 0 or 1.
- $\mathsf{OWR.Update}(\mathsf{sk}_{i-1}) \to \mathsf{sk}_i$. The update algorithm takes in a secret key sk_{i-1} and produces a new secret key sk_i for the *same* public key.

Correctness. The OWR scheme satisfies correctness if for any polynomial $q = q(\kappa)$, it holds that for all $i \in \{0, 1, \ldots, q\}$, $\mathsf{OWR.Verify}(\mathsf{pk}, \mathsf{sk}_i) = 1$, where $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{OWR.Gen}(1^\kappa)$, and $\mathsf{sk}_{i+1} \leftarrow \mathsf{OWR.Update}(\mathsf{sk}_i)$.

Security. We define continual leakage security for one-way relations in terms of the following game between a challenger and an attacker. We let κ denote the security parameter, and the parameter μ controls the amount of leakage allowed.

Setup Phase. The game begins with a setup phase. The challenger calls $\mathsf{OWR.Gen}(1^\kappa)$ to create the initial secret key sk_0 and public key pk. It gives pk to the attacker. No leakage is allowed in this phase.

Query Phase. In this phase, the attacker launches a polynomial number of leakage queries. Each time, say in the ith query, the attacker provides an efficiently computable leakage function f_i whose output is at most μ bits, and the challenger chooses randomness r_i, updates the secret key from sk_{i-1} to sk_i, and gives the attacker the leakage response ℓ_i. In the CLR model, the leakage attack is applied on a single secret key, and the leakage response $\ell_i = f_i(\mathsf{sk}_{i-1})$. In the 2CLR model, the leakage attack is applied on consecutive two secret keys, i.e., $\ell_i = f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)$. In the model of CLR with leakage on key updates, the leakage attack is applied on the current secret key and the randomness used for updating the secret key, i.e., $\ell_i = f_i(\mathsf{sk}_{i-1}, r_i)$.

Recovery Phase. The attacker outputs some sk^*. The attacker wins the game if $\mathsf{OWR.Verify}(\mathsf{pk}, \mathsf{sk}^*) = 1$. We define the success probability of the attacker in this game as $\Pr[\mathsf{OWR.Verify}(\mathsf{pk}, \mathsf{sk}^*) = 1]$.

Definition 11. (*Continual Leakage Resilience*) We say a one-way relation scheme is μ-CLR secure (respectively, μ-2CLR secure, or μ-CLR secure with leakage on key updates) if any PPT attacker only has a negligible advantage (negligible in κ) in the above game.

6.2. *Construction Based on LIRR*

Leakage-Indistinguishable Re-randomizable Relation

In [21], Dodis et al introduce a new primitive, *leakage-indistinguishable re-randomizable relation (LIRR)*, and show that this primitive can be used to construct OWR in the CLR model where the adversary is allowed to leak on the secret key in each round of leakage attack. LIRR allows one to sample two types of secret keys: "good" keys and "bad" keys. Both types of keys look valid and are acceptable by the verification procedure, but they are produced in very different ways. In fact, given the ability to produce good keys, it is hard to produce any bad key and vice-versa. On the other hand, even though the two types of keys are very different, they are hard to distinguish from each other. More precisely, given the ability to produce both types of keys, and μ bits of leakage on a "challenge" key of an unknown type (good or bad), it is hard to come up with a new key of the same type. More formally, a LIRR consists of PPT algorithms $(\mathsf{Setup}, \mathsf{SampG}, \mathsf{SampB}, \mathsf{Update}, \mathsf{Verify}, \mathsf{isGood})$ with the following syntax:

- $(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\kappa)$: This algorithm returns a public key pk, a "good" sampling key s_G, a "bad" sampling key s_B, and a distinguishing trapdoor dk.
- $\mathsf{sk}_G \leftarrow \mathsf{SampG}_{\mathsf{pk}}(s_G)$ and $\mathsf{sk}_B \leftarrow \mathsf{SampB}_{\mathsf{pk}}(s_B)$: These algorithms sample good/bad secret keys using good/bad sampling keys, respectively. We omit the subscript pk when clear from context.
- $b \leftarrow \mathsf{isGood}(\mathsf{pk}, \mathsf{sk}, \mathsf{dk})$: This algorithm uses dk to distinguish good secret keys sk from bad ones.
- $\mathsf{sk}_i \leftarrow \mathsf{Update}(\mathsf{sk}_{i-1})$ and $b \leftarrow \mathsf{Verify}(\mathsf{pk}, \mathsf{sk})$: These two algorithms have the same syntax as in the definition of OWR in the CLR model.

Definition 12. We say $(\mathsf{Setup}, \mathsf{SampG}, \mathsf{SampB}, \mathsf{Update}, \mathsf{Verify}, \mathsf{isGood})$ is a μ leakage-indistinguishable re-randomizable relation (LIRR) if it satisfies the following properties:

Correctness: If $(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\kappa)$, $\mathsf{sk}_G \leftarrow \mathsf{SampG}(s_G)$, $\mathsf{sk}_B \leftarrow \mathsf{SampB}(s_B)$ then $\Pr\left[\begin{array}{c}\mathsf{Verify}(\mathsf{pk}, \mathsf{sk}_G) = 1 \bigwedge \mathsf{isGood}(\mathsf{pk}, \mathsf{sk}_G, \mathsf{dk}) = 1 \\ \bigwedge \mathsf{Verify}(\mathsf{pk}, \mathsf{sk}_B) = 1 \bigwedge \mathsf{isGood}(\mathsf{pk}, \mathsf{sk}_B, \mathsf{dk}) = 0\end{array}\right] = 1 - \mathsf{negl}(\kappa)$

Re-Randomization: We require that $(\mathsf{pk}, s_G, \mathsf{sk}_0, \mathsf{sk}_1) \overset{c}{\approx} (\mathsf{pk}, s_G, \mathsf{sk}_0, \mathsf{sk}'_1)$ where $(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Setup}()$, $\mathsf{sk}_0 \leftarrow \mathsf{SampG}(s_G)$ and $\mathsf{sk}_1 \leftarrow \mathsf{Update}(\mathsf{sk}_0)$, $\mathsf{sk}'_1 \leftarrow \mathsf{SampG}(s_G)$

Hardness of Bad Keys: Given s_G, it is hard to produce a valid "bad key". Formally, for any PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\kappa), \mathsf{sk}^* \leftarrow \mathcal{A}(\mathsf{pk}, s_G) : \\ \mathsf{Verify}(\mathsf{pk}, \mathsf{sk}^*) = 1 \bigwedge \mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = 0\end{array}\right] \leq \mathsf{negl}(\kappa)$$

Hardness of Good Keys: Given s_B, it is hard to produce a valid "good key". Formally, for any PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\kappa), \mathsf{sk}^* \leftarrow \mathcal{A}(\mathsf{pk}, s_B) : \\ \mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = 1\end{array}\right] \leq \mathsf{negl}(\kappa)$$

μ Leakage Indistinguishability: Given both sampling keys s_G, s_B, and μ bits of leakage on a secret key sk (which is either good or bad), it is hard to produce a secret key sk^* which is in the same category as sk. Formally, for any PPT adversary $\mathcal{A}$, we have $|\Pr[\mathcal{A} \text{ wins }] - 1/2| \leq \mathsf{negl}(\kappa)$ in the following game:

- The challenger chooses $(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\kappa)$ and gives pk, s_G, s_B to $\mathcal{A}$. The challenger chooses a random bit $b \in \{0, 1\}$. If $b = 1$, then it samples $\mathsf{sk} \leftarrow \mathsf{SampG}(s_G)$, and otherwise, it samples $\mathsf{sk} \leftarrow \mathsf{SampB}(s_B)$.
- The adversary $\mathcal{A}$ can make up to q queries in total to the leakage oracle
- The adversary outputs sk^* and wins if $\mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = b$.

6.2.1. *Construction*

A μ LIRR can be used to construct a μ-2CLR-secure OWR, as follows:

– $\mathsf{Gen}(1^\kappa)$: Sample $(\mathsf{pk}, s_G, \cdot, \cdot) \leftarrow \mathsf{Setup}(1^\kappa)$, $\mathsf{sk} \leftarrow \mathsf{SampG}(s_G)$ and output $(\mathsf{pk}, \mathsf{sk})$
– $\mathsf{Update}(\cdot)$, $\mathsf{Verify}(\cdot, \cdot)$: Same as for LIRR

Note that the CLR-OWR completely ignores the bad sampling algorithm SampB, the "bad" sampling key s_B, the distinguishing algorithm isGood, and the distinguishing key dk of the LIRR. These are only used in the argument of security. Moreover, the "good" sampling key s_G is only used as an intermediate step during key generation to sample the secret key sk, but is never explicitly stored afterward.

Theorem 11. *Given any 2μ-LIRR scheme, the construction above is a μ-2CLR-secure OWR.*

Proof. The proof is very similar to that in [21]. To prove the theorem statement, we develop a sequence of games. □

Game $\mathcal{H}_0$: This is the original μ-2CLR Game as Definition 11. The adversary is allowed to apply leakage function on consecutive two secret keys in each round of leakage attack. *Games $\mathcal{H}_{0.i}$ - $\mathcal{H}_1$*: Let q be the total number of leakage rounds for which $\mathcal{A}$ runs. We define the Games $\mathcal{H}_{0.i}$ for $i = 0, 1, \ldots, q$ as follows. The challenger initially samples $(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\kappa)$, and $\mathsf{sk}_0 \leftarrow \mathsf{SampG}(s_G)$ and gives pk to $\mathcal{A}$. The game then proceeds as before with many leakage rounds, except that the secret keys in rounds $j \leq i$ are generated as $\mathsf{sk}_j \leftarrow \mathsf{SampG}(s_G)$, independently of all previous rounds, and in the rounds $j > i$, they are generated as $\mathsf{sk}_j \leftarrow \mathsf{Update}(\mathsf{sk}_{j-1})$. Note that Game $\mathcal{H}_{0.0}$ is the same as Game $\mathcal{H}_0$, and we define Game $\mathcal{H}_1$ to be the same as Game $\mathcal{H}_{0.q}$.

Claim 12. *For $i = 1, \ldots, q$, it holds that* $|\Pr[\mathcal{A} \text{ wins } |\mathcal{H}_{0.(i-1)}] - \Pr[\mathcal{A} \text{ wins } |\mathcal{H}_{0.i}]| \leq \mathsf{negl}(\kappa)$

Proof. We use the re-randomization property to argue that, for $i = 1, \ldots, q$, the winning probability of $\mathcal{A}$ is the same in Game $\mathcal{H}_{0.(i-1)}$ as in Game $\mathcal{H}_{0.i}$, up to negligible factors. We construct a reduction $\mathcal{B}$, with input $(\mathsf{pk}, s_G, \mathsf{sk}', \mathsf{sk}'')$. Here $\mathsf{sk}' \leftarrow \mathsf{SampG}(s_G)$, and sk'' is sampled based on randomly chosen b: if $b = 1$, then $\mathsf{sk}'' \leftarrow \mathsf{SampG}(s_G)$ and if $b = 0$, then $\mathsf{sk}'' \leftarrow \mathsf{Update}(\mathsf{sk}')$.

More concretely, the reduction $\mathcal{B}$ emulates a copy of $\mathcal{A}$ internally. In addition, $\mathcal{B}$ emulates the view for $\mathcal{A}$: For all $j < i$, $\mathcal{B}$ generates $\mathsf{sk}_j \leftarrow \mathsf{SampG}(s_G)$, and for all $j > i + 1$, $\mathcal{B}$ generates $\mathsf{sk}_j \leftarrow \mathsf{Update}(\mathsf{sk}_{j-1})$; $\mathcal{B}$ sets $\mathsf{sk}_i := \mathsf{sk}'$ and $\mathsf{sk}_{i+1} := \mathsf{sk}''$. Upon receiving a leakage query f_j from $\mathcal{A}$, the reduction $\mathcal{B}$ returns $f_j(\mathsf{sk}_{j-1}, \mathsf{sk}_j)$ to $\mathcal{A}$.

If $\mathcal{B}$'s challenger uses sk'' which is generated through SampG then that corresponds to the view of $\mathcal{A}$ in Game $\mathcal{H}_{0.i}$ and if sk'' is generated through Update, then corresponds to Game $\mathcal{H}_{0.(i-1)}$. Therefore, if $\mathcal{A}$ is able to distinguish the two worlds, then $\mathcal{B}$ is able to break the re-randomization property. □

Game $\mathcal{H}_2$: Game $\mathcal{H}_2$ is the same as Game $\mathcal{H}_1$, except the winning condition: Now the adversary only wins if, at the end, it outputs sk^* such that $\mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = 1$.

Claim 13. $|\Pr[\mathcal{A} \text{ wins } |\mathcal{H}_1] - \Pr[\mathcal{A} \text{ wins } |\mathcal{H}_2]| \leq \mathsf{negl}(\kappa)$

Proof. The winning probability of $\mathcal{A}$ in Game $\mathcal{H}_2$ is at least that of Game $\mathcal{H}_1$ minus the probability that sk^* satisfies $\mathsf{Verify}(\mathsf{pk}, \mathsf{sk}^*) = 1 \wedge \mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = 0$. However, since the entire interaction between the challenger and the adversary in games $\mathcal{H}_1, \mathcal{H}_2$ can be simulated using (pk, s_G), we can use the "hardness of bad keys" property to argue that the probability of the above happening is negligible. Therefore, the probability of $\mathcal{A}$ winning in Game $\mathcal{H}_2$ is at least that of Game $\mathcal{H}_1$, up to negligible factors. □

Games $\mathcal{H}_{2.i}$ - $\mathcal{H}_3$: Let q be the total number of leakage rounds for which $\mathcal{A}$ runs. We define the Games $\mathcal{H}_{2.i}$ for $i = 0, 1, \ldots, q$ as follows. The challenger initially samples $(\mathsf{pk}, s_G, s_B, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and gives pk to $\mathcal{A}$. The game then proceeds as before with many leakage rounds, except that the secret keys in rounds $j \leq i$ are generated as $\mathsf{sk}_j \leftarrow \mathsf{SampB}(s_B)$, and in the rounds $j > i$, they are generated as $\mathsf{sk}_j \leftarrow \mathsf{SampG}(s_G)$. Note that Game $\mathcal{H}_{2.0}$ is the same as Game $\mathcal{H}_2$, and we define Game $\mathcal{H}_3$ to be the same as Game $\mathcal{H}_{2.q}$.

Claim 14. *For* $i = 1, \ldots, q$, *it holds that* $|\Pr[\mathcal{A} \text{ wins } |\mathcal{H}_{2.(i-1)}] - \Pr[\mathcal{A} \text{ wins } |\mathcal{H}_{2.i}]| \leq \mathsf{negl}(\kappa)$.

Proof. We use the 2μ-leakage indistinguishability property to argue that, for $i = 1, \ldots, q$, the winning probability of $\mathcal{A}$ is the same in Game $\mathcal{H}_{2.(i-1)}$ as in Game $\mathcal{H}_{2.i}$, up to negligible factors. We construct a reduction $\mathcal{B}$, with input (pk, s_G, s_B) and with leakage access to sk. Here sk is sampled based on randomly chosen b: if $b = 1$, then $\mathsf{sk} \leftarrow \mathsf{SampG}(s_G)$ and if $b = 0$, then $\mathsf{sk} \leftarrow \mathsf{SampB}(s_B)$.

More concretely, the reduction $\mathcal{B}$ emulates a copy of $\mathcal{A}$ internally. In addition, $\mathcal{B}$ emulates the view for $\mathcal{A}$: In each leakage round $j < i$, $\mathcal{B}$ uses s_B to generate $\mathsf{sk}_j \leftarrow \mathsf{SampB}(s_B)$; and in round $j > i$, $\mathcal{B}$ uses s_G to generate $\mathsf{sk}_j \leftarrow \mathsf{SampG}(s_G)$. Upon receiving a leakage query f_j from $\mathcal{A}$ in leakage round j, if $j < i - 1$, $\mathcal{B}$ returns $f_j(\mathsf{sk}_{j-1}, \mathsf{sk}_j)$ to $\mathcal{A}$; if $j = i - 1$, $\mathcal{B}$ defines $\hat{f}_j = f_j(\mathsf{sk}_{j-1}, \cdot)$, and then applies $\hat{f}_j$ on sk, and returns $\hat{f}_j(\mathsf{sk})$ to $\mathcal{A}$; if $j = i$, $\mathcal{B}$ defines $\hat{f}_j = f_j(\mathsf{sk}_j, \cdot)$, and then applies $\hat{f}_j$ on sk, and returns $\hat{f}_j(\mathsf{sk})$ to $\mathcal{A}$; if $j > i$, $\mathcal{B}$ returns $f_j(\mathsf{sk}_{j-1}, \mathsf{sk}_j)$ to $\mathcal{A}$. At the end, $\mathcal{B}$ outputs the value sk^* output by $\mathcal{A}$. Note that $\mathcal{B}$ made queries $\hat{f}_{i-1}$ and $\hat{f}_i$, which are 2μ bits in total.

If $\mathcal{B}$'s challenger uses a good key then that corresponds to the view of $\mathcal{A}$ in Game $\mathcal{H}_{2.i}$ and a bad key corresponds to Game $\mathcal{H}_{2.(i-1)}$. Therefore, letting b be the bit used by $\mathcal{B}$'s challenger, we have:

$$\begin{aligned}
|\Pr[\mathcal{B} \text{ wins }] - 1/2| &= \left|\Pr[\mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = b] - 1/2\right| \\
&= 1/2 \cdot \Big|\Pr[\mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = 1 | b = 1] \\
&\quad - \Pr[\mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = 1 | b = 0]\Big| \\
&= 1/2 \cdot \left|\Pr[\mathcal{A} \text{ wins } |\mathcal{H}_{2.(i-1)}] - \Pr[\mathcal{A} \text{ wins } |\mathcal{H}_{2.i}]\right|
\end{aligned}$$

Claim 15. $\Pr[\mathcal{A} \text{ wins } |\mathcal{H}_3] \leq \mathsf{negl}(\kappa)$

Proof. We now argue that probability of $\mathcal{A}$ winning Game $\mathcal{H}_3$ is negligible, by the "hardness of good keys". Notice that $\mathcal{A}$'s view in Game $\mathcal{H}_3$ can be simulated entirely just given (pk, s_B). Therefore, there is a PPT algorithm which, given (pk, s_B) as inputs,

Construction $\mathsf{OWR} = \mathsf{OWR}.\{\mathsf{Gen}, \mathsf{Update}, \mathsf{Verify}\}$

Here $\mathcal{E} = \mathcal{E}.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Update}\}$ is a PKE scheme, and $p(\cdot)$ is some polynomial.

Key Generation: $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{OWR.Gen}(1^\kappa)$
- Compute $(\mathsf{pk}^{\mathcal{E}}, \mathsf{sk}_0^{\mathcal{E}}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$.
- Choose $p = p(\kappa)$ random messages $m_1, \ldots, m_p$ from the message space of $\mathcal{E}$.
- For $i \in [p]$, compute a random encryption $e_i = \mathcal{E}.\mathsf{Enc}(m_i)$.
- Output public key $\mathsf{pk} = (\mathsf{pk}^{\mathcal{E}}, (m_1, e_1), \ldots, (m_p, e_p))$ and secret key $\mathsf{sk}_0 = \mathsf{sk}_0^{\mathcal{E}}$.

Key Update: $\mathsf{sk}_{i+1} \leftarrow \mathsf{OWR.Update}(\mathsf{sk}_i)$
Set $\mathsf{sk}_{i+1} \leftarrow \mathcal{E}.\mathsf{Update}(\mathsf{sk}_i)$.

Verification: $b \leftarrow \mathsf{OWR.Verify}(\mathsf{sk}', \mathsf{pk})$
Upon receiving key pair $(\mathsf{sk}', \mathsf{pk})$, parse $\mathsf{pk} = (\mathsf{pk}^{\mathcal{E}}, (m_1, e_1), \ldots, (m_p, e_p))$.
If for all $i \in [p]$, $\mathcal{E}.\mathsf{Dec}(\mathsf{sk}', e_i) = m_i$, then set $b := 1$; otherwise, set $b := 0$.

Fig. 16. Transformation of PKE to OWR.

can run Game $\mathcal{H}_3$ with $\mathcal{A}$ and output sk^* such that $\mathsf{isGood}(\mathsf{pk}, \mathsf{sk}^*, \mathsf{dk}) = 1$ whenever $\mathcal{A}$ wins. So the probability of $\mathcal{A}$ winning in Game $\mathcal{H}_3$ is negligible. □

By the hybrid argument, the probability of $\mathcal{A}$ winning in Game $\mathcal{H}_0$ is at most that of $\mathcal{A}$ winning in Game $\mathcal{H}_3$, up to negligible factors. That is, $\Pr[\mathcal{A} \text{ wins } |\mathcal{H}_0] \leq \mathsf{negl}(\kappa)$. Therefore, since the latter is negligible, the former must be negligible as well, which concludes the proof of the theorem. □

Based on the result in [21], we have the following corollary.

Corollary 1. *Fix a constant $K \geq 1$, and assume that the K-linear assumption holds in the base groups of some pairing. Then, for any constant $\epsilon > 0$, there exists a μ-2CLR-secure OWR scheme with relative-leakage $\frac{\mu}{|\mathsf{sk}|} \geq \frac{1}{2(K+1)} - \epsilon$.*

6.3. *A Generic Construction Based on PKE*

In this section, we describe a generic construction of CLR-secure OWR (resp., 2CLR secure and CLR with leakage on key updates) from CLR secure PKE (resp., 2CLR secure, and CLR with leakage on key updates). A OWR requires verification of the relation to be deterministic; but a PKE does not necessarily give a OWR because there might not be a deterministic way to check the key pair $(\mathsf{pk}, \mathsf{sk})$ of a PKE. Here we present a way to check the key pair of a PKE *deterministically*, so that one can use PKE to construct OWR.

Theorem 12. *Let $\mathcal{E}$ be a public key encryption scheme secure in the model of CLR (respectively, of 2CLR, and of CLR with leakage on key updates) with leakage rate ρ, then for appropriate choice of polynomial $p(\cdot)$, the one-way relation scheme* OWR *in Fig. 16 is secure in the model of CLR (respectively, of 2CLR, and of CLR with leakage on key updates) with leakage rate ρ.*

Proof. (*Sketch.*) A well-known result from learning theory known as *Occam's Razor* (see, for example, Kearns and Vazirani [40], Theorem 2.1)[11] says that if a class of circuits has size $|\mathcal{C}|$ and a circuit $C \in \mathcal{C}$ agrees with a target circuit $C^* \in \mathcal{C}$ on $\text{poly}(\log(|\mathcal{C}|), 1/\epsilon, \log(1/\delta))$ number of random inputs, then with probability $1-\delta$, C agrees with C^* over the uniform distribution with probability $1-\epsilon$. In the following, we will always set $\log(1/\delta) \geq \kappa$ and so $\delta \leq 1/2^\kappa$.

Assume we have an adversary $\mathcal{A}$ breaking the security of the one-way relation, we use it to construct an adversary $\mathcal{A}'$ breaking the security of the encryption scheme $\mathcal{E}$. The class $\mathcal{C}$ consists of the circuits $\mathcal{E}.\mathsf{Dec}(\widetilde{\mathsf{sk}}, \cdot)$ for all possible sk. Clearly, $\log(|\mathcal{C}|) = |\mathsf{sk}| = \text{poly}(\kappa)$. Now, C corresponds to the circuit $\mathcal{E}.\mathsf{Dec}(\mathsf{sk}', \cdot)$, where sk' is the secret key submitted by $\mathcal{A}$ such that $\mathsf{OWR}.\mathsf{Verify}(\mathsf{sk}', \mathsf{pk}) = 1$. Furthermore, C^* is the circuit $\mathcal{E}.\mathsf{Dec}(\mathsf{sk}, \cdot)$, where sk is a real secret key. Note that C and C^* agree on $p(\kappa) = \text{poly}(\log(|\mathcal{C}|), 1/\epsilon, 1/\log(\delta))$ random inputs, since $\mathcal{E}.\mathsf{Verify}(\mathsf{sk}', \mathsf{pk}) = 1$. Thus, we are guaranteed that with probability $1-\delta$ over choice of input/output pairs (m_i, e_i) in $\mathsf{pk}, \mathsf{sk}'$ decrypts correctly on a fresh random input with probability $1-\epsilon$. We are now ready to define the adversary $\mathcal{A}'$.

$\mathcal{A}'$ internally instantiates $\mathcal{A}$, while participating externally in a leakage (resp., on consecutive two keys, and on both key and update) on encryption scheme $\mathcal{E}$. Specifically, $\mathcal{A}'$ does the following:

- Upon receiving $\mathsf{pk}^{\mathcal{E}}$ from the external experiment, do the following:
  - Choose $p = p(\kappa)$ random messages $m_1, \ldots, m_p$ from the message space of $\mathcal{E}$.
  - For $i \in [p]$, compute a random encryption $e_i = \mathcal{E}.\mathsf{Enc}(m_i)$.
  - Output public key $\mathsf{pk} = (\mathsf{pk}^{\mathcal{E}}, (m_1, e_1), \ldots, (m_p, e_p))$ to the internal adversary $\mathcal{A}$. Note that secret key $\mathsf{sk}_0 = \mathsf{sk}_0^{\mathcal{E}}$ is a correctly distributed secret key for this pk.
- Whenever $\mathcal{A}$ submits a leakage query f, $\mathcal{A}'$ submits the same query to its external challenger who applies it to the secret key (resp., on consecutive two keys, and on both key and update) and forwards the answer to $\mathcal{A}$.
- Finally, $\mathcal{A}$ submits sk' to $\mathcal{A}'$. If there exists $i \in [p]$ such that $\mathcal{E}.\mathsf{Dec}(\mathsf{sk}', e_i) \neq m_i$, then $\mathcal{A}'$ outputs random b'.
- Otherwise, $\mathcal{A}'$ chooses two independent, uniformly random messages m_0, m_1 and submits to its external challenger.
- $\mathcal{A}'$ then receives the challenge ciphertext c^*.
- $\mathcal{A}'$ computes $m^* = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}', c^*)$. If $m^* = m_0$, $\mathcal{A}'$ outputs 0. Otherwise, $\mathcal{A}'$ outputs 1.

Note that $\mathcal{A}'$ perfectly simulates $\mathcal{A}$'s view in the OWR game. Therefore, it is not hard to see that if $\mathcal{A}$ succeeds with probability $p_1 = p_1(\kappa) \geq 8/2^\kappa$, then $\mathcal{A}'$ succeeds with probability $1/2 \cdot (1-p_1) + (p_1 - \delta)(1-\epsilon)$. For $\epsilon \leq 1/7$, we have that $(p_1 - \delta)(1-\epsilon) \geq 3p_1/4$. Thus, $\mathcal{A}'$ succeeds with probability $1/2 + p_1/4$ and obtains advantage $\mathsf{Adv}_{\mathcal{A}',\mathcal{E}} = p_1/4$. This, in turn, implies that $\mathcal{A}$ must succeed with negligible $p_1(\kappa)$ probability, since otherwise we contradict the security of $\mathcal{E}$. □

[11] We note that the statement of Occam's Razor theorem in [40] is for the case of Boolean functions. However, the analysis can be easily extended to the non-Boolean case.

7. Continual Leakage Resilience for Digital Signatures

In the previous section, we extended the techniques of Dodis et al. [21] to construct μ-2CLR-secure one-way relations. Dodis et al. [21] showed how to construct continual leakage-resilient signature schemes from one-way relations secure against continual leakage. In this section, we extend their techniques and to construct μ-2CLR-secure signature schemes from μ-2CLR-secure one-way relations. Finally, combining our resulting μ-2CLR-secure signature scheme with Theorems 6 and 7, we obtain a continual leakage-resilient signature scheme with leakage on update (but no leakage on the randomness used for signing).

See Sects. 2.2 and 3.4 for the formal definition of continual (consecutive) leakage resilience for digital signature schemes.

7.1. *NIZK and True-Simulation Extractability*

Dodis et al. [21] constructed CLR-secure signature based on CLR-secure OWR and another primitive named *true-simulation extractable (tSE)* NIZK. We here recall the syntax and security properties of NIZK. We note that the definitions below are taken from [21] for completeness.

Let R be an NP relation on pairs (y, x) with corresponding language $L_R = \{y \mid \exists x \text{ s. t. } (y, x) \in R\}$. A NIZK argument for a relation R consists of four PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ with syntax:

- $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa)$: Creates a common reference string (CRS) and a trapdoor key to the CRS.
- $\pi \leftarrow \mathsf{Prove}_{\mathsf{crs}}(y, x)$: Creates an argument that $y \in L_R$.
- $\mathsf{Sim}_{\mathsf{crs}}(y, \mathsf{tk})$: Creates a simulated argument that $y \in L_R$.
- $b \leftarrow \mathsf{Verify}_{\mathsf{crs}}(y, \pi)$: Verifies whether or not the argument π is correct.

For the sake of clarity, we write $\mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim}$ without the crs in the subscript when the crs can be inferred from the context.

Definition 13. We say that $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ are a NIZK argument system for the relation R if the following three properties hold.

Completeness: For any $(y, x) \in R$, if $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa), \pi \leftarrow \mathsf{Prove}(y, x)$, then $Verify(y, \pi) = 1$.

Soundness: For any PPT adversary $\mathcal{A}$, $\Pr[\mathsf{Verify}(y, \pi^*) = 1 \wedge y \notin L_R : (\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa), (y, \pi^*) \leftarrow \mathcal{A}(\mathsf{crs})] \leq \mathsf{negl}(1^\kappa)$

Composable Zero Knowledge: For any PPT adversary $\mathcal{A}$, we have $\Pr[\mathcal{A}\text{wins}] - 1/2 \leq \mathsf{negl}(1^\kappa)$ in the following game:

- The challenger samples $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa)$ and gives $(\mathsf{crs}, \mathsf{tk})$ to $\mathcal{A}$.
- The adversary $\mathcal{A}$ chooses $(y, x) \in R$ and gives these to the challenger.
- The challenger samples $\pi_0 \leftarrow \mathsf{Prove}(y, x), \pi_1 \leftarrow \mathsf{Sim}(y, \mathsf{tk}), b \leftarrow \{0, 1\}$ and gives π_b to $\mathcal{A}$.
- The adversary $\mathcal{A}$ outputs a bit b' and wins if $b' = b$.

Definition 14. (*True-Simulation Extractability* [22]) Let $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ be an NIZK argument for an NP relation R, satisfying the completeness, soundness and zero-knowledge properties. We say that NIZK is *true-simulation extractable (tSE)*, if:

- Apart from outputting a CRS and a trapdoor key, Setup also outputs an extraction key: $(\mathsf{crs}, \mathsf{tk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\kappa)$.
- There exists a PPT algorithm $\mathsf{Ext}_{\mathsf{ek}}$ such that for all $\mathcal{A}$ we have $\Pr[\mathcal{A}\text{wins }] \leq \mathsf{negl}(1^\kappa)$ in the following game:
  1. The challenger runs $(\mathsf{crs}, \mathsf{tk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\kappa)$ and gives crs to $\mathcal{A}$.
  2. $\mathcal{A}^{\mathcal{SIM}_{\mathsf{tk}}()}$ is given access to a simulation oracle $\mathcal{SIM}_{\mathsf{tk}}()$, which it can adaptively access. A query to the simulation oracle consists of a pair (y, x). The oracle checks if $(y, x) \in R$. If true, it ignores x and outputs a simulated argument $\mathsf{Sim}_{\mathsf{tk}}(y)$. Otherwise, the oracle outputs $\perp$.
  3. $\mathcal{A}$ outputs a pair (y^*, σ^*), and the challenger runs $x^* \leftarrow \mathsf{Ext}_{\mathsf{ek}}(y^*, \sigma^*)$.

$\mathcal{A}$ wins if $(y, x^*) \notin R, \mathsf{Verify}(y^*, \sigma^*) = 1$, and y^* was not part of a query to the simulation oracle.

7.2. *CLR Signatures with Leakage on Key Updates from OWR*

Next we recall Dodis et al's construction and then show that actually their construction is 2CLR secure if the underlying OWR is 2CLR secure. We then combine the above with our generic transformation from Sect. 3 to obtain CLR-secure signatures with leakage on key updates.

In the following, $\mathsf{OWR} := (\mathsf{OWR.Gen}(1^\kappa), \mathsf{OWR.Update}(\mathsf{sk}))$ is a 2CLR-secure one-way relation and $\mathsf{NIZK} := (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ is a is tSE-NIZK for the relation

$$R = \{(y, x) \mid y = (\mathsf{pk}, m), x = \mathsf{sk} \text{ s.t. } \mathsf{Verify}(\mathsf{pk}, \mathsf{sk}) = 1\}.$$

Although input m seems useless in the above relation, looking ahead, m will play the role of the message to be signed. Note that we have the important property that when the message m changes, the statement $y = (\mathsf{pk}, m)$ also changes.

- $\mathsf{SIG.Gen}(1^\kappa)$: Output $(\mathsf{vk}, \mathsf{sk})$ where $\mathsf{vk} = (\mathsf{pk}, \mathsf{crs})$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{OWR.Gen}(1^\kappa)$ and $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\kappa)$.
- $\mathsf{SIG.Sign}_{\mathsf{sk}}(m)$: Output $\sigma \leftarrow \mathsf{NIZK.Prove}((\mathsf{pk}, m), \mathsf{sk})$.
- $\mathsf{SIG.Verify}_{\mathsf{vk}}(m, \sigma)$: Output $b := \mathsf{NIZK.Verify}(\mathsf{pk}, m), \sigma)$.
- $\mathsf{SIG.Update}(\mathsf{sk})$: Output $\mathsf{OWR.Update}(\mathsf{sk})$.

Theorem 13. *If one-way relation* OWR *is* μ*-2CLR secure and* NIZK *is true-simulation extractable, then the above signature scheme is* μ*-2CLR secure.*

Proof. The proof here is very similar to that in [21]. We prove the above theorem through a sequence of games. □

Game $\mathcal{H}_0$: This is the original μ-2CLR game described in Definition 3, in which signing queries are answered honestly by running $\sigma \leftarrow \mathsf{NIZK.Prove}((\mathsf{pk}, m), \mathsf{sk})$ and $\mathcal{A}$ wins if she produces a valid forgery (m^*, σ^*).

Game $\mathcal{H}_1$: In this game, the signing queries are answered by generating simulated arguments, i.e., $\sigma \leftarrow \mathsf{NIZK.Sim}_{\mathsf{tk}}(\mathsf{pk}, m)$. Games $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable by the zero-knowledge property of NIZK. Here the simulated arguments given to $\mathcal{A}$ as answers to signing queries are always of true statements.

Game $\mathcal{H}_2$: In this game, we modify the winning condition so that the adversary only wins if it produces a valid forgery (m^*, σ^*) and the challenger is able to extract a valid secret key sk^* for pk from (m^*, σ^*). That is, $\mathcal{A}$ wins if both $\mathsf{SIG.Verify}(m^*, \sigma^*) = 1$ and $\mathsf{OWR.Verify}(\mathsf{pk}, \mathsf{sk}^*) = 1$, where $\mathsf{sk}^* \leftarrow \mathsf{NIZK.Ext}_{\mathsf{ek}}((\mathsf{pk}, m^*), \sigma^*)$. The winning probability of $\mathcal{A}$ in Game $\mathcal{H}_2$ is at least that of Game $\mathcal{H}_1$ minus the probability that $\mathsf{NIZK.Verify}((\mathsf{pk}, m^*), \sigma^*) = 1$ and $\mathsf{OWR.Verify}(\mathsf{pk}, \mathsf{sk}^*) = 0$. By the true-simulation extractability of the argument NIZK, we know that this probability is negligible. Therefore, the winning probability of $\mathcal{A}$ in Game $\mathcal{H}_2$ differs from that in Game $\mathcal{H}_1$ by a negligible amount.

We have shown that the probability that $\mathcal{A}$ wins in Game $\mathcal{H}_0$ is the same as that in Game $\mathcal{H}_2$, up to negligible factors. We now argue that the probability that $\mathcal{A}$ wins in Game $\mathcal{H}_2$ is negligible, which proves that the probability that $\mathcal{A}$ wins in Game $\mathcal{H}_0$ is negligible as well. To prove that the probability that $\mathcal{A}$ wins in Game $\mathcal{H}_2$ is negligible, we assume otherwise and show that there exists a PPT algorithm $\mathcal{B}$ that breaks the μ-2CLR security of OWR. On input pk, $\mathcal{B}$ generates $(\mathsf{crs}, \mathsf{tk}, \mathsf{ek}) \leftarrow \mathsf{NIZK.Setup}(1^\kappa)$ and emulates $\mathcal{A}$ on input $\mathsf{vk} = (\mathsf{crs}, \mathsf{pk})$. In each leakage round, $\mathcal{B}$ answers $\mathcal{A}$'s leakage queries using the leakage oracle and answers signing queries m_i by creating simulated arguments $\sigma_i \leftarrow \mathsf{NIZK.Sim}_{\mathsf{tk}}(\mathsf{pk}, m_i)$. When $\mathcal{A}$ outputs her forgery (m^*, σ^*), $\mathcal{B}$ runs $\mathsf{sk}^* \leftarrow \mathsf{NIZK.Ext}_{\mathsf{ek}}((\mathsf{pk}, m^*), \sigma^*)$ and outputs sk^*. Notice that $\Pr[\mathcal{B} \text{ wins }] = \Pr[\mathcal{A} \text{ wins }]$, so that if $\Pr[\mathcal{A} \text{ wins }]$ is non-negligible then $\mathcal{B}$ breaks the μ-consecutive two-key security of OWR. We therefore conclude that the probability that $\mathcal{A}$ wins in Game $\mathcal{H}_2$ is negligible. This concludes the proof of the theorem. □

Based on the result in [21], we have the following corollary.

Corollary 2. *Fix a constant $K \geq 1$, and assume that the K-linear assumption holds in the base groups of some pairing. Then, for any constant $\epsilon > 0$, there exists a μ-2CLR-secure signature scheme with relative-leakage $\frac{\mu}{|\mathsf{sk}|} \geq \frac{1}{2(K+1)} - \epsilon$.*

Acknowledgements

We thank the anonymous reviewers for their insightful comments, which greatly improved the presentation of this work.

A. The Connection Between CLR and Obfuscation

In Sects. 5.1 and 5.2, we demonstrated that we can use obfuscation to fortify any public key encryption scheme with leakage resilience, if the leakage is one-time and bounded. Here, we seek to generalize the approach, using obfuscation to achieve security against continual leakage. As compared with the result of Sect. 4, where we show that program obfuscation can strengthen one specific encryption scheme to provide leakage resilience, the intention here is to explore what can be done generically. That is, our aim is to achieve continual leakage resilience starting from any PKE scheme and diO. Unfortunately, we fall somewhat short, requiring that the underlying encryption scheme posses certain specific properties, which we describe below. Although we demonstrate an instantiation of such a scheme from the Decision-Linear assumption [7], we believe that the most interesting open question left by our study of the relationship between leakage resilience and program obfuscation is to construct such an encryption scheme directly from any PKE scheme and diO. We include this section, in large part, with the hope of highlighting the missing piece.

Recall that in the previous construction, the secret key consists of a dummy ciphertext. An initial idea for how to achieve (consecutive) CLR is to refresh the secret key by *re-randomizing* the ciphertext. Specifically, in the real construction, the ciphertext in the secret key just encrypts zeros and is re-randomized, but in the proof it contains the PRF output and the underlying plaintext is refreshed from round to round. However, since the underlying plaintext changes from round to round, we run into some technical difficulties. Specifically, an adversary who knows the underlying secret key for the ciphertext embedded in the construction's secret key will be able to distinguish consecutive hybrids. On the other hand, an adversary who does not know the secret key for the ciphertext will not be able to produce a correctly distributed obfuscated circuit to be placed in the public key. To resolve this, the idea is to use an encryption scheme with special properties: The challenge ciphertext remains semantically secure while at the same time, the adversary can efficiently *simulate* a decryption oracle which either successfully decrypts the submitted ciphertext, or indicates that the submitted ciphertext is a re-randomization of the challenge ciphertext. Note that this notion is a strengthening of the notion of re-randomizable RCCA (relaxed CCA) security. Specifically, we define a new special "diO-compatible" notion of "relaxed" [15] or "controlled" malleability CCA security for re-randomizable encryption [17]. We then show how to realize our new notion from the Decision-Linear (DLIN) assumption in bilinear groups, following [17]. Our resulting continual leakage-resilient scheme presented in Sect. A.4 combines this assumption with diO.

A.1. *Re-randomizable Encryption*

A *re-randomizable encryption scheme* is a tuple of algorithms $\mathsf{RPKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ defined as follows. First, the triple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a standard encryption scheme. Second,

$$(\mathrm{PK}, \mathrm{SK}, c_1, c_2) \approx_c (\mathrm{PK}, \mathrm{SK}, c_1, c_2')$$

where

$$(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)\ ;\ c_1 \leftarrow \mathsf{Enc}(\mathrm{PK}, m),$$
$$c_2 \leftarrow \mathsf{Enc}(\mathrm{PK}, m)\ ;\ c_2' \leftarrow \mathsf{ReRand}(\mathrm{PK}, c)\ .$$

A.2. *diO-Compatible RCCA Encryption*

Intuitively, an RCCA encryption scheme [15] is like a CCA-secure encryption scheme that allows *replay attacks*. We define a special type of RCCA encryption scheme that we call *diO compatible*, inspired by the definition of *controlled malleability* of Chase et al. [17]. The intuition for diO-compatible RCCA encryption is that the security proof allows for placing the secret key of the encryption scheme in an obfuscation, while arguing about semantic security of a challenge ciphertext. Specifically, the way this is achieved is by requiring that the diO-compatible RCCA encryption have a specific hybrid structure for proving security. In each hybrid, there is an efficient algorithm which simulates the decryption oracle. Moreover, indistinguishability of consecutive hybrids reduces either to the security of an underlying primitive, or reduces to the fact that, even given the code of the simulated decryption oracle, the attacker cannot find a *distinguishing input*. A detailed definition follows.

Syntax. A *diO-compatible RCCA encryption scheme* is a tuple of algorithms $\mathsf{RCCA} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SimEnc}, \mathsf{Dec})$ where

- Algorithm $\mathsf{Gen}(1^\kappa)$ outputs keys $(\mathrm{PK}, (\mathrm{SK}_1, \mathrm{SK}_2), \tau_{\mathsf{sim}})$.
- Algorithm $\mathsf{Enc}(\mathrm{PK}, m)$ outputs a ciphertext c.
- Algorithm $\mathsf{SimEnc}(\tau_{\mathsf{sim}}, m)$ outputs a ciphertext c_{sim}.
- Algorithm $\mathsf{Dec}_1(\mathrm{SK}_1, c)$ outputs a message value in $\{m, \perp\}$.
- Algorithm $\mathsf{Dec}_2(\mathrm{SK}_2, c)$ outputs a message value in $\{m, \perp, \mathsf{SimFlag}\}$.

Correctness. For correctness, we require that $\Pr[\mathsf{Dec}(\mathrm{SK}_1, \mathsf{Enc}(\mathrm{PK}, m)) = m] = 1$ where $(\mathrm{PK}, (\mathrm{SK}_1, \mathrm{SK}_2), \tau_{\mathsf{sim}})$ is output by $\mathsf{Gen}(1^\kappa)$. (This can be relaxed to allow negligible probability of failure.)

Security. Intuitively, our encryption scheme must remain secure, even when the adversary is given an obfuscation of the decryption key (using diO security). This is hard to argue generically, since we cannot say for sure what might be revealed about the key by such an obfuscation. By using a two-key system and a NIZK, we enable ourselves to transition through a sequence of hybrids that allow us to change the content of the plaintext while still claiming indistinguishability. (A similar method appears in the work of Garg et al. [29].) More specifically, we require three security properties that are formally described below. The first property says that real ciphertexts are indistinguishable from simulated ones, even when given SK_1; this allows us to transition to using simulated ciphertexts, even when the adversary is given an obfuscation containing SK_1. The third property says that, given a simulated ciphertext encrypting m, it is hard to construct a new ciphertext that decrypts to $m' \neq m$ under SK_1, but decrypts to something other than m' under SK_2. This allows us to switch the key in the obfuscation from SK_1 to SK_2 under the argument of diO security. Finally, the second security property ensures indistinguishability of simulated ciphertexts even when given SK_2, which allows us to modify plaintext values even while giving the adversary the obfuscated key. The formal definition follows. We

note that in the definition of simulation soundness, we use $\mathsf{SimFlag}$ to denote a Boolean value, which is intended to indicate that a ciphertext was simulated. [12]

– **Indistinguishability of simulated ciphertexts from real ciphertexts:** For any efficient adversary A and message m

$$A(\mathrm{PK}, \mathsf{Enc}(\mathrm{PK}, m), \mathrm{SK}_1) \overset{c}{\approx} A(\mathrm{PK}, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m), \mathrm{SK}_1) \ .$$

– **Indistinguishability of simulated ciphertexts under chosen plaintext attack:** For any efficient adversary A, and any message pair (m_0, m_1),

$$A(\mathrm{PK}, \mathrm{SK}_2, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m_0)) \overset{c}{\approx} A(\mathrm{PK}, \mathrm{SK}_2, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m_1)) \ .$$

– **Simulation soundness:** For any efficient A and message m the probability that the following experiment outputs 1 is negligible: $(\mathrm{PK}, (\mathrm{SK}_1, \mathrm{SK}_2), \tau_{\mathsf{sim}}) \leftarrow \mathsf{Gen}(1^\kappa)$; $c \leftarrow \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m)$; $c^* \leftarrow A(\mathrm{SK}_1, \mathrm{SK}_2, c)$ Return 1 if either

1. $\mathsf{Dec}_2(\mathrm{SK}_2, c^*) = \mathsf{SimFlag} \bigwedge \mathsf{Dec}_1(\mathrm{SK}_1, c^*) \neq m$
2. $\mathsf{Dec}_2(\mathrm{SK}_2, c^*) \neq \mathsf{SimFlag} \bigwedge \mathsf{Dec}_2(\mathrm{SK}_2, c^*) \neq \mathsf{Dec}_1(\mathrm{SK}_1, c^*)$

where the probability is taken over the randomness used in key generation, SimEnc, and by A when computing c^*.

Re-randomizability. We say that a diO-compatible RCCA-secure encryption scheme is *re-randomizable* if it has an additional algorithm ReRand defined analogously to re-randomizable encryption.

A.3. *Construction of diO-Compatible RCCA Re-randomizable PKE*

In our extension to continual leakage, we need a diO-compatible RCCA-secure *re-randomizable* scheme. We will show that the controlled-malleable (CM) CCA encryption scheme of Chase et al. [17] instantiates this notion based on the Decision-Linear assumption [7]. Compared to their security analysis, we prove something stronger since we require an RCCA scheme of a particular "diO-compatible" form, but we use the *same* RCCA construction of [17] and what we require follows by observing that their scheme possesses the additional properties we require.

We begin by defining non-interactive proof systems, non-interactive zero-knowledge (NIZK) proofs of knowledge and malleable proof systems. We then define the additional security properties (controlled-malleable simulation sound extractability), required for the construction of Chase et al. [17]. Our definitions closely follow those in [17].

Definition 15. (*Non-interactive proof systems*) A set of algorithms $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ constitute a non-interactive (NI) proof system for an efficient relation R with associated language L_R if completeness and soundness below are satisfied. A NI proof system

[12] We note that the presented conditions only refer to *single-message security*, which is all that is needed in our applications. More generally they could provide the adversary many challenge encryptions (say, via appropriate encryption oracles).

is extractable if, in addition, the extractability property below is satisfied. A NI proof system is zero knowledge (NIZK) if the zero-knowledge property is satisfied. A NIZK proof system that is also extractable constitutes a non-interactive zero-knowledge proof of knowledge (NIZKPoK) system.

1. Completeness. For all $\sigma_{crs} \leftarrow \mathsf{CRSSetup}(1^\kappa)$ and $(x, w) \in R$, $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$ for all proofs $\pi \leftarrow \mathcal{P}(\sigma_{crs}, x, w)$.
2. Soundness. For all PPT $\mathcal{A}$, and for $\sigma_{crs} \leftarrow \mathsf{CRSSetup}(1^\kappa)$, the probability that $\mathcal{A}(\sigma_{crs})$ outputs (x, π) such that $x \notin L$ but $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$, is negligible.
3. Extractability. There exists a polynomial-time extractor algorithm $E = (E_1, E_2)$ such that $E_1(1^\kappa)$ outputs $(\sigma_{ext}, \tau_{\mathsf{ext}})$ and $E_2(\sigma_{ext}, \tau_{\mathsf{ext}}, x, \pi)$ outputs a value w such that (1) a σ_{ext} output by $E_1(1^\kappa)$ is indistinguishable from σ_{crs} output by $\mathsf{CRSSetup}(1^\kappa)$; (2) for all PPT $\mathcal{A}$, the probability that $\mathcal{A}(\sigma_{ext}, \tau_{\mathsf{ext}})$ (where $(\sigma_{ext}, \tau_{\mathsf{ext}}) \leftarrow E_1(1^\kappa)$ outputs (x, π) such that $\mathcal{V}(\sigma_{ext}, x, \pi) = 1$ and $R(x, E_2(\sigma_{ext}, \tau_{\mathsf{ext}}, x, \pi)) = 0$, is negligible.
4. Zero knowledge. There exists a polynomial-time simulator algorithm $S = (S_1, S_2)$ such that $S_1(1^\kappa)$ outputs $(\sigma_{sim}, \tau_{\mathsf{sim}})$ and $S_2(\sigma_{sim}, \tau_{\mathsf{sim}}, x)$ outputs a value π_s such that for all $(x, w) \in R$ and PPT adversaries $\mathcal{A}$, the following two interactions are indistinguishable: in the first, we compute $\sigma_{crs} \leftarrow \mathsf{CRSSetup}(1^\kappa)$ and give $\mathcal{A}$ σ_{crs} and oracle access to $\mathcal{P}$ (where $\mathcal{P}$ will output $\perp$ on input (x, w) such that $(x, w) \notin R$); in the second we compute $(\sigma_{sim}, \tau_{\mathsf{sim}})$ and give $\mathcal{A}$ σ_{sim} and oracle access to $S(\sigma_{sim}, \tau_{\mathsf{sim}}, \cdot, \cdot)$, where, on input (x, w), S outputs $S_2(\sigma_{sim}, \tau_{\mathsf{sim}}, x)$ if $(x, w) \in R$ and $\perp$ otherwise.

Definition 16. (*Malleable non-interactive proof system*) Let $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ be a non-interactive proof system for a relation R. Let $\mathcal{T}$ be an allowable set of transformations for R. Then this proof system is *malleable with respect to* $\mathcal{T}$ if there exists an efficient algorithm ZKEval that on input $(\sigma_{crs}, T, \{x_i, \pi_i\})$, where $T \in \mathcal{T}$ is an n-ary transformation, and $\mathcal{V}(\sigma_{crs}, x_i, \pi_i) = 1$ for all i, $1 \leq i \leq n$, outputs a valid proof π for the statement $x = T_x(\{x_i\})$ (i.e., a proof π such that $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$).

Definition 17. (*Controlled-malleable simulation sound extractability*) Let $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ be a NIZKPoK system for an efficient relation R, with a simulator (S_1, S_2) and an extractor (E_1, E_2). Let $\mathcal{T}$ be an allowable set of unary transformations for the relation R such that membership in $\mathcal{T}$ is efficiently testable. Let SE_1 be an algorithm that on input 1^κ outputs $(\sigma_{crs}, \tau_{\mathsf{sim}}, \tau_{\mathsf{ext}})$ such that $(\sigma_{crs}, \tau_{\mathsf{sim}})$ is distributed identically to the output of S_1. Let $\mathcal{A}$ be given, and consider the following game:

- Step 1. $(\sigma_{crs}, \tau_{\mathsf{sim}}, \tau_{\mathsf{ext}}) \leftarrow SE_1(1^\kappa)$.
- Step 2. $(x, \pi) \leftarrow \mathcal{A}^{S_2(\sigma_{crs}, \tau_{\mathsf{sim}}, \cdot)}(\sigma_{crs}, \tau_{\mathsf{ext}})$.
- Step 3. $(w, x', T) \leftarrow E_2(\sigma_{crs}, \tau_{\mathsf{ext}}, x, \pi)$.

We say that the NIZKPoK satisfies controlled-malleable simulation sound extractability if for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that the probability (over the choices of SE_1, $\mathcal{A}$ and S_2) that $\mathcal{V}(\sigma_{crs}, x, \pi) = 1$ and $(x, \pi) \notin Q$ (where Q is the set of queried statements and their responses) and either (1) $w \neq \perp$ and $(x, w) \notin R$; (2) $(x', T) \neq (\perp, \perp)$ and either $x' \notin T_x(x')$ or $T \notin \mathcal{T}$; or (3) $(w, x', T) = (\perp, \perp, \perp)$ is at most $\mathsf{negl}(\kappa)$.

For simplicity, we additionally require ($\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}$) to be "same-string" NIZK (see [20]), which means that σ_{crs} generated by SE_1 is identically distributed to σ_{crs} generated by $\mathsf{CRSSetup}$. We point out in the analysis below where this property is used.

For our purposes, we require proof systems for statements of the form "I know the message and randomness corresponding to public key and ciphertext pair (pk, c), for an underlying re-randomizable encryption scheme." Given a proof corresponding to a particular (pk, c), we would like to use malleability to construct proofs for $(pk, \widetilde{c})$, where $\widetilde{c}$ is a re-randomization of c. Therefore, we require an NIZKPoK with controlled malleability (CM) with respect to the class $\mathcal{T}$, where $\mathcal{T}$ corresponds to the set of transformations that take as input (pk, c) and output $(pk, \widetilde{c})$, where $\widetilde{c}$ is a re-randomization of c. We discuss instantiations of such CM-NIZK below.

Our instantiation of diO-compatible RCCA follows the construction of CM-CCA encryption of [17]. Details follow.

The Instantiation of diO-Compatible RCCA based on [17]. We assume a re-randomizable IND-CPA-secure encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ and a cm-NIZK scheme $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$, with simulator SE_1, for the relation R such that $((pk, c), (m, r)) \in R$ iff $c := \mathsf{Enc}'(pk, m; r)$ and for malleability class $\mathcal{T}$, where $\mathcal{T}$ is the set of transformations corresponding to re-randomization of the ciphertext c. For our construction we have $\mathsf{RCCA} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SimEnc}, \mathsf{Dec})$ as follows:

- $\mathsf{Gen}(1^\kappa)$: Run $(pk', sk') \leftarrow \mathsf{Gen}'(1^\kappa)$ and $(\sigma_{crs}, \tau_{\mathsf{sim}}, \tau_{\mathsf{ext}}) \leftarrow SE_1(1^\kappa)$; set the public key pk of the RCCA encryption scheme to $\mathrm{PK} := (pk', \sigma_{crs})$, the secret key SK_1 of the RCCA encryption scheme to $\mathrm{SK}_1 := (\mathrm{PK}, sk')$, and the secret key SK_2 of the RCCA encryption scheme to $\mathrm{SK}_2 := (\mathrm{PK}, \tau_{\mathsf{ext}})$. Output $(\mathrm{PK}, \mathrm{SK}_1, \mathrm{SK}_2, \tau_{\mathsf{sim}})$.
- $\mathsf{Enc}(\mathrm{PK}, m)$: Parse $\mathrm{PK} = (pk', \sigma_{crs})$; then compute $c' \leftarrow \mathsf{Enc}'(\mathsf{pk}', m)$ and $\pi \leftarrow \mathcal{P}(\sigma_{crs}, (pk', c'), m)$ (i.e., a proof of knowledge of the value inside c') and output $c := (c', \pi)$.
- $\mathsf{SimEnc}(\mathrm{PK}, \tau_{\mathsf{sim}}, m)$: Parse $\mathrm{PK} = (pk', \sigma_{crs})$, then compute $c'_{\mathsf{sim}} \leftarrow \mathsf{Enc}'(\mathsf{pk}', m)$ and $\pi_{\mathsf{sim}} \leftarrow S_2(\sigma_{crs}, \tau_{\mathsf{sim}}, (pk', c'_{\mathsf{sim}}))$ a *simulated* proof of plaintext knowledge as π_{sim}; it outputs ciphertext $c := (c'_{\mathsf{sim}}, \pi_{\mathsf{sim}})$.
- $\mathsf{Dec}_1(\mathrm{SK}_1, c)$: First parse $\mathrm{SK}_1 = (\mathrm{PK}, sk')$, $\mathrm{PK} := (pk', \sigma_{crs})$ and $c := (c', \pi)$; now check that $\mathcal{V}(\sigma_{crs}, (pk', c'), \pi) = 1$. If not abort and output $\perp$. Otherwise, compute and output $m = \mathsf{Dec}'(sk', c')$.
- $\mathsf{Dec}_2(\mathrm{SK}_2, c)$: First parse $\mathrm{SK}_1 = (\mathrm{PK}, sk')$, $\mathrm{PK} := (pk', \sigma_{crs})$ and $c := (c', \pi)$; now check that $\mathcal{V}(\sigma_{crs}, (pk', c'), \pi) = 1$. If not abort and output $\perp$. Otherwise, compute $((m, r), (pk', c'), T) \leftarrow E_2(\sigma_{crs}, \tau_{\mathsf{ext}}, x, \pi)$. If $(m, r) = \perp$, output $\mathsf{SimFlag}$. Otherwise, output m.

diO Compatibility of the Scheme. We go through the required security properties and sketch the argument for why each of them hold.

– **Indistinguishability of simulated ciphertexts from real ciphertexts:** For any efficient adversary A and message m

$$A(\text{PK}, \mathsf{Enc}(\text{PK}, m), \text{SK}_1) \overset{c}{\approx} A(\text{PK}, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m), \text{SK}_1) \ .$$

This follows from the "same-string" NIZK property of $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ and the fact that simulated proofs and real proofs under the cm-NIZK scheme $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ are indistinguishable.

– **Indistinguishability of simulated ciphertexts under chosen plaintext attack:** For any efficient adversary A, and any message pair (m_0, m_1),

$$A(\text{PK}, \text{SK}_2, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m_0)) \overset{c}{\approx} A(\text{PK}, \text{SK}_2, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m_1)) \ .$$

Since S_2 takes as input σ_{crs}, τ_{sim}, (pk', c'_{sim}), the output of $\mathsf{SimEnc}(\tau_{\mathsf{sim}}, m_b)$, $b \in \{0, 1\}$ can be computed given $(\sigma_{crs}, \tau_{\mathsf{sim}}, c'^{b}_{\mathsf{sim}})$, where c'^{b}_{sim} is an encryption of m_b under pk'. Therefore, due to the definitions of PK, SK_2, it is sufficient to show that

$$(\sigma_{crs}, \tau_{\mathsf{ext}}, \tau_{\mathsf{sim}}, c'^{0}_{\mathsf{sim}}) \overset{c}{\approx} (\sigma_{crs}, \tau_{\mathsf{ext}}, \tau_{\mathsf{sim}}, c'^{1}_{\mathsf{sim}}).$$

This follows from the fact that encryptions of m_0 and encryptions of m_1 under pk' are indistinguishable, even given σ_{crs}, τ_{ext}, and τ_{sim}.

– **Simulation soundness:** For any efficient A and message m the probability that the following experiment outputs 1 is negligible: $(\text{PK}, (\text{SK}_1, \text{SK}_2), \tau_{\mathsf{sim}}) \leftarrow \mathsf{Gen}(1^\kappa)$; $c \leftarrow \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m)$; $c^* \leftarrow A(\text{SK}_1, \text{SK}_2, c)$ Return 1 if

1. $\mathsf{Dec}_2(\text{SK}_2, c^*) = \mathsf{SimFlag} \bigwedge \mathsf{Dec}_1(\text{SK}_1, c^*) \neq m$
2. $\mathsf{Dec}_2(\text{SK}_2, c^*) \neq \mathsf{SimFlag} \bigwedge \mathsf{Dec}_2(\text{SK}_2, c^*) \neq \mathsf{Dec}_1(\text{SK}_1, c^*)$

where the probability is taken over the randomness used in key generation, SimEnc, and by A when computing c^*.

This follows from the controlled-malleable simulation sound extractability of the proof system $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$, which holds even given τ_{ext} (which is contained in SK_2). Specifically, parse $c := (c', \pi)$ and $c^* := (c^{*\prime}, \pi^*)$ and note that if $\mathsf{Dec}_2(\text{SK}_2, c^*) = \mathsf{SimFlag}$ then it must be the case that $\mathcal{V}(\sigma_{crs}, (pk', c^{*\prime}), \pi^*) = 1$ and $((m^*, r^*), (pk', c^{*\prime}), T) \leftarrow E_2(\sigma_{crs}, \tau_{\mathsf{ext}}, x, \pi)$ is such that $(m^*, r^*) = \perp$. But by the controlled-malleable simulation sound extractability of $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$, if the above occurs then it must be the case that $(pk', c^{*\prime}) \in T_{(pk', c')}(pk', c^{*\prime})$ and $T \in \mathcal{T}$. This means that $c^{*\prime}$ is a re-randomization of c' and so $\mathsf{Dec}_1(\text{SK}_1, c^*) = \mathsf{Dec}'(sk', c^{*\prime}) = m$. On the other hand, if $\mathsf{Dec}_2(\text{SK}_2, c^*) \neq \mathsf{SimFlag}$ then if $\mathcal{V}(\sigma_{crs}, (pk', c^{*\prime}), \pi^*) = 0$ both $\mathsf{Dec}_2(\text{SK}_2, c^*)$ and $\mathsf{Dec}_1(\text{SK}_1, c^*)$ output $\perp$. Otherwise, it must be the case that $\mathcal{V}(\sigma_{crs}, (pk', c^{*\prime}), \pi^*) = 1$,

$((m^*, r^*), (pk', c^{*\prime}), T) \leftarrow E_2(\sigma_{crs}, \tau_{\mathsf{ext}}, x, \pi)$ is such that $(m^*, r^*) \neq \perp$ and $\mathsf{Dec}_2(\text{SK}_2, c^*)$ outputs m^*. But by the controlled-malleable simulation sound extractability of $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$, if the above occurs then it must be the case that $((m^*, r^*),$

$(pk', c^{*\prime})) \in R$, which means that $\mathsf{Dec}_1(\mathrm{SK}_1, c^*) = \mathsf{Dec}'(sk', c^{*\prime}) = m^* = \mathsf{Dec}_2(\mathrm{SK}_2, c^*)$.

Concrete parameters. As in [17], the underlying re-randomizable RCCA public key encryption scheme can be instantiated with the Decision-Linear-based encryption scheme of Boneh, Boyen, and Shacham [7] (which is re-randomizable via exponentiation). The NIZK proof system can be the same as in [17]: Combine a Groth-Sahai proof of plaintext knowledge (which is itself re-randomizable and supports exponentiation malleability of the underlying statement) along with the signature scheme from Abe et al. [1]. A ciphertext in the resulting diO-compatible RCCA-secure re-randomizable PKE scheme will contain a constant number of group elements, although this constant is large. A benefit of our abstractions is that better constructions of re-randomizable RCCA-secure public key encryption and corresponding NIZK proof system will lead to improved parameters for our continual leakage-resilient PKE scheme.

A.4. *Our 2CLR PKE Construction*

To guarantee security in the presence of continual leakage, we modify our construction in two ways. First, we strengthen the security of the encryption scheme used to encrypt $\mathsf{ct}_{\mathsf{dummy}}$, requiring that it provide relaxed CCA security (RCCA) [15]. Recall that such encryption schemes allow users to re-randomize ciphertexts, while guaranteeing the ciphertexts are otherwise secure against chosen ciphertext attacks.

The other change that we make to the scheme of Sect. 5.2 comes up in the proof of security. Because we are leaking over multiple rounds, storing t^* in a list no longer suffices. After enough rounds, the value will be fully recovered by the adversary, and he will distinguish neighboring hybrids. To fix this, we instead store random values whose inner product yields the challenge point, along with a hash of the challenge. In our proof of security, the most interesting hybrids are Hybrid 4, where use RCCA security, Hybrid 5, where we reduce to diO, and Hybrid 8, where we use the fact that inner product is a good two-source (and, therefore, strong) extractor. The other hybrids are fairly straightforward (Figs. 17, 18, 19).

Theorem 14. *Assume*

- RCCA *is a diO-compatible RCCA-secure re-randomizable PKE with ciphertexts of length* $L_{\mathsf{ct}}(\kappa, L_{\mathsf{msg}})$ *for* L_{msg}*-bit messages and security parameter* κ.
- SIG *is a strong existentially unforgeable digital signature scheme with signatures of length* $L_{\mathsf{sig}}(\kappa, L_{\mathsf{msg}})$ *for* L_{msg}*-bit messages and security parameter* κ.
- PRF *is a puncturable pseudorandom function* $\{0,1\}^\kappa \times \{0,1\}^\rho \to \{0,1\}^{L_{\mathsf{msg}}}$ *for some* $\rho = \rho(\kappa) = \omega(\kappa^2)$.
- G *is a pseudorandom generator* $\{0,1\}^\kappa \to \{0,1\}^\rho$.
- diO *is a differing-inputs obfuscator for circuits in this scheme.*
- $\mathcal{H}$ *is a family of collision-resistant hash functions with output size* κ *bits.*

Encryption Scheme $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec}, \mathcal{E}.\mathsf{Update})$

Key Generation: $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$

Compute the following:

- $(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_{\mathsf{RCCA}}, \mathrm{SK}_2, \tau_{\mathsf{sim}}) \leftarrow \mathsf{RCCA.Gen}(1^\kappa)$,
- $(\mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SIG.Gen}(1^\kappa)$,
- $k \leftarrow \mathsf{PRF.Gen}(1^\kappa)$.
- $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{td}, 0^{2\kappa+\rho+L_{\mathsf{msg}}})$.
- $\mathsf{ct}_{\mathsf{dummy}} \leftarrow \mathsf{RCCA.Enc}(\mathrm{PK}_{\mathsf{RCCA}}, \sigma || 0^{2\kappa+\rho+L_{\mathsf{msg}}}; r_0)$, where $r_0 \leftarrow \{0,1\}^\kappa$.

Let C_k be the circuit described in Figure 18, and compute $C_{\mathsf{Enc}} \leftarrow \mathsf{diO}(C_k)$.
Let $\mathsf{keys} = \{\mathrm{SK}_{\mathsf{RCCA}}, k, \mathsf{vk}\}$, let C_{keys} be the circuit in Figure 19, and compute $C_{\mathsf{Dec}} \leftarrow \mathsf{diO}(C_{\mathsf{keys}})$.
Output $\mathsf{pk} = (\mathrm{PK}_{\mathsf{RCCA}}, C_{\mathsf{Enc}}, C_{\mathsf{Dec}})$ and $\mathsf{sk}_0 = (\mathsf{ct}_{\mathsf{dummy}})$.

Encryption: $c \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m)$

On input message $m \in \{0,1\}^{L_{\mathsf{msg}}}$, sample $r \leftarrow \{0,1\}^\kappa$, and output $c = (G(r), C_{\mathsf{Enc}}(r) \oplus m)$, where G is some fixed pseudorandom generator.

Decryption: $\hat{m} \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}_i, c)$

In round i, on input ciphertext $c = (t, v)$, compute $y := C_{\mathsf{Dec}}(\mathsf{sk}_i, t)$.
If $y \neq \perp$, output $\hat{m} = y \oplus v$. Otherwise, output $\hat{m} = \perp$.

Key Update: $\mathsf{sk}_i \leftarrow \mathcal{E}.\mathsf{Update}(\mathsf{sk}_{i-1})$

In round i, on input secret key sk_{i-1}, randomly choose $r_i \leftarrow \{0,1\}^\kappa$, compute and output $\mathsf{sk}_i \leftarrow \mathsf{RCCA.ReRand}(\mathrm{PK}_{\mathsf{RCCA}}, \mathsf{sk}_{i-1}; r_i)$.

Fig. 17. Continual leakage-resistant encryption scheme, $\mathcal{E}$.

Internal (hardcoded) state: k.

On input: r

- Output $z = \mathsf{PRF.Eval}(k, G(r))$, where G is some fixed pseudorandom generator.

Fig. 18. Program C_k. This program is obfuscated and placed in the public key to be used for encryption.

Internal (hardcoded) state: $\mathsf{keys} = \{k, \mathsf{vk}, \mathrm{SK}_{\mathsf{RCCA}}\}$.

On input: $\mathsf{ct}_{\mathsf{dummy}}, t$

- Compute $(\sigma', m) = \mathsf{RCCA.Dec}_1(\mathrm{SK}_{\mathsf{RCCA}}, \mathsf{ct}_{\mathsf{dummy}})$.
- If $\mathsf{SIG.Verify}(\sigma', m; \mathsf{vk}) = 0$ output $\perp$.
- Output $z = \mathsf{PRF.Eval}(k, t)$.

Fig. 19. Program C_{keys}. This program is obfuscated and placed in the public key. It is used during decryption.

Then $\mathcal{E}$ is L-2CLR where

$$\frac{L}{|\mathsf{sk}|} = \frac{(1/6 - o(1))\rho}{L_{\mathsf{ct}}\Big(\kappa, 2\kappa + \rho + L_{\mathsf{msg}} + L_{\mathsf{sig}}(\kappa, 2\kappa + \rho + L_{\mathsf{msg}})\Big)}$$

We choose a diO-compatible RCCA-secure re-randomizable PKE with $L_{\mathsf{ct}}(\kappa, L_{\mathsf{msg}}) = c_1 \cdot L_{\mathsf{msg}}$, for some constant c_1 and a signature scheme with $L_{\mathsf{sig}}(\kappa, L_{\mathsf{msg}}) = o(L_{\mathsf{msg}})$. Setting $\rho = \rho(\kappa) = \omega(\kappa^2)$ yields an encryption scheme for messages of length $\Theta(\kappa)$ with constant leakage rate $c_1/6 - o(1)$.

Note that a diO-compatible RCCA-secure re-randomizable PKE is achieved by the Chase et al. [17] scheme, as argued in the previous section. Additionally, signature schemes with the required property (that $L_{\mathsf{sig}}(\kappa, L_{\mathsf{msg}}) = o(L_{\mathsf{msg}})$) can be achieved using the well-known "hash-and-sign" paradigm (see, for example, [38]) and can be constructed assuming the existence of collision-resistant hash functions.

It may seem puzzling as to why the leakage rate seems to depends on the length of the message being encrypted, L_{msg}. In general, it is true that the length of the message being encrypted should not affect the leakage rate. However, in our scheme we are using a "one-time-pad encryption" paradigm where C_{Enc} generates randomness (t, y) and y is then used for a one-time-pad encryption of the message, so that the final ciphertext is $(t, v := y \oplus m)$. Note that this means that the length of y is the same as the length of the message (L_{msg}). C_{Dec} then reverses this process, by taking as input the secret key ($\mathsf{ct}_{\mathsf{dummy}}$) and t (from the ciphertext) and returning the corresponding y. In our proof, we use an information theoretic argument for one of the steps, which requires that, for a fixed t^*, the output of C_{Dec} (of length L_{msg} bits) is uniform random, even conditioned on the leakage from the secret key. Clearly, in this argument, the entropy of the output of C_{Dec} must be coming from the entropy of the secret key. The entropy of the secret key, in turn, comes from the fact that, in the hybrid argument, we switch from an encryption of an all-0 string to an encryption of a random string, y^*. This means that the secret key must encrypt a message of length at least L_{msg} bits (even under 0 bits of leakage) and so clearly the length of the secret key must depend on the length of the message, L_{msg}. If, instead of using one-time-pad encryption, we used a computational variant in our construction, we would eliminate the dependence between the secret key length and the message length. However, this would further complicate the proof, requiring additional steps in the hybrid argument. Therefore, for simplicity, we assume a simple one-time-pad-based encryption.

In order to prove Theorem 14, we prove (in Lemma 16) that even under leakage, it is hard for any PPT adversary $\mathcal{A}$ to distinguish the output of the PRF y from uniform random. Given this, Theorem 14 follows immediately.

In fact, we will prove a slightly stronger lemma, by allowing $\mathcal{O}$ to leak on two consecutive keys in any given round. By combining this property with the results from Sect. 3, we prove that this construction achieves security when the adversary is allowed to leak on updates. More specifically, in the lemma below, in round i, we allow the adversary to specify a leakage function $f_i(\cdot, \cdot)$, and $\mathcal{O}$ returns $f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)$.

Lemma 16. *For every* PPT *leaking adversary $\mathcal{A}$, who is given oracle access to a leakage oracle $\mathcal{O}$ and may leak at most L bits of the secret key, there exist random variables $\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}_0, \ldots, \widetilde{\mathsf{sk}}_n$ such that:*

$$\left(\mathsf{pk}, t^*, y, \{f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk})\right)$$
$$\overset{c}{\approx} \left(\widetilde{\mathsf{pk}}, U_\rho, U_{L_{\mathsf{msg}}}, \{f_i(\widetilde{\mathsf{sk}}_{i-1}, \widetilde{\mathsf{sk}}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widetilde{\mathsf{pk}})\right)$$

where $y = C_{\mathsf{Dec}}(\mathsf{ct}_{\mathsf{dummy}}, t = G(r))$, *n is the number of key update rounds requested by* $\mathcal{A}$, *and the distributions are taken over coins of* $\mathcal{A}$ *and choice of* $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $\mathsf{sk}_i \leftarrow \mathcal{E}.\mathsf{Update}(\mathsf{sk}_{i-1})$, r *and choice of* $\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}_0, \ldots, \widetilde{\mathsf{sk}}_n, w$, *respectively.*

Proof. We prove the lemma via the following sequence of hybrids:

Hybrid 0: This hybrid is identical to the real game.

Let $\mathcal{D}^{\mathcal{A}}_{H_0}$ denote the distribution $(\mathsf{pk}, t, y, \{f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}))$ as in the left side of Lemma 16.

Hybrid 1: This hybrid is the same as Hybrid 0 except we replace pseudorandom $t = G(r)$ in the challenge ciphertext with uniform random $t^* \leftarrow \{0,1\}^\rho$. Let $\mathcal{D}^{\mathcal{A}}_{H_1}$ denote the distribution $(\mathsf{pk}, t^*, y, \{f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}))$ where $y = C_{\mathsf{Dec}}(\mathsf{sk}_n, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, choice of $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, $\mathsf{sk}_i \leftarrow \mathcal{E}.\mathsf{Update}(\mathsf{sk}_{i-1})$, t^* as described above. □

Claim 16. *For every* PPT *adversary* $\mathcal{A}$,

$$\mathcal{D}^{\mathcal{A}}_{H_0} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_1}.$$

Proof. The proof follows from the security of the PRG used in C_{Enc}. We refer the reader to the proof of Claim 6. We note that the reduction holds even if the adversary were given all sk_i in full. □

Hybrid 2: This hybrid is the same as Hybrid 1 except we replace the key k used in C_{Enc} with a punctured key, $\widetilde{k} = \mathsf{PRF}.\mathsf{Punct}(k, t^*)$. We denote the resulting public key by pk'. Let $\mathcal{D}^{\mathcal{A}}_{H_2}$ denote the distribution $\left(\mathsf{pk}', t^*, y, \{f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$ where $y = C_{\mathsf{Dec}}(\mathsf{sk}_n, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \mathsf{sk}_0, \ldots, \mathsf{sk}_n), t^*$ as described above.

Claim 17. *For every* PPT *adversary* $\mathcal{A}$,

$$\mathcal{D}^{\mathcal{A}}_{H_1} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_2}.$$

Proof. The proof follows from the security of the obfuscation used in C_{Enc}. We refer the reader to the proof of Claim 7. We note again that the reduction holds even if $\mathcal{A}$ is given all sk_i in full. □

Hybrid $3^{(j)}$: We define a sequence of n hybrids, where n is the number of key update rounds requested by the adversary. In Hybrid $3^{(j)}$, in the first j update rounds, instead of refreshing $\mathsf{ct}_{\mathsf{dummy}}$ by computing $\mathsf{ct}_{\mathsf{dummy}} = \mathsf{RCCA}.\mathsf{ReRand}(\mathrm{PK}_{\mathsf{RCCA}}, \mathsf{ct}_{\mathsf{dummy}})$, we replace the secret key with a fresh ciphertext: $\mathsf{ct}_{\mathsf{dummy}} = \mathsf{RCCA}.\mathsf{Enc}(\mathrm{PK}_{\mathsf{RCCA}}, \sigma \| 0^{2\kappa+\rho+L_{\mathsf{msg}}})$. For rounds $i > j$, the secret key is still refreshed through a re-randomization. We denote the resulting set of secret keys by $\{\mathsf{sk}_0^{(3|j)}, \ldots, \mathsf{sk}_n^{(3|j)}\}$. Let $\mathcal{D}^{\mathcal{A}}_{H_{3|j}}$ denote the distribution $\left(\mathsf{pk}', t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(3|j)}, \mathsf{sk}_i^{(3|j)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$ where

$y = C_{\mathsf{Dec}}(\ \mathsf{sk}_n^{(3|j)},\ t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \mathsf{sk}_0^{(3|j)}, \ldots, \mathsf{sk}_n^{(3|j)}), t^*$ as described above.

Claim 18. *For every* PPT *adversary* $\mathcal{A}$*, and every* $j \in [n]$

$$\mathcal{D}^{\mathcal{A}}_{H_{3|j-1}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{3|j}}.$$

Proof. The proof is through a reduction to the property that the distribution of fresh ciphertexts and the distribution of re-randomized ciphertexts are indistinguishable, even given the secret key sk_j. The reduction adversary $\mathcal{S}$ receives $(\mathsf{pk}, \mathsf{sk}_j)$ from his challenger and uses them to generate the public keys for $\mathcal{E}$, along with the secret keys $\{\mathsf{sk}_0 = \mathsf{RCCA.Enc}(\mathrm{PK}_{\mathsf{RCCA}}, \sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}}), \ldots, \mathsf{sk}_{j-1} = \mathsf{RCCA.Enc}(\mathrm{PK}_{\mathsf{RCCA}}, \sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}})\}$. He submits ciphertext sk_{j-1} along with the underlying plaintext value as his challenge and receives c^* which is either a re-randomization sk_{j-1}, or a fresh encryption. He computes sk_{j+1} by re-randomizing his challenge ciphertext, and for $i \in \{j+1, \ldots, n\}$, he computes sk_i by re-randomizing sk_{i-1}. These ciphertexts are distributed either identically to those in Hybrid $3^{(j-1)}$ or to those in Hybrid $3^{(j)}$ and can be used by $\mathcal{S}$ to perfectly simulate the responses to $\mathcal{A}$'s leakage queries. It follows that the advantage of $\mathcal{S}$ is the same as the advantage of distinguishing $\mathcal{D}^{\mathcal{A}}_{H_{3|j-1}}$ from $\mathcal{D}^{\mathcal{A}}_{H_{3|j}}$. □

Hybrid 4: In this hybrid step, we again change the update phase. Instead of replacing $\mathsf{ct}_{\mathsf{dummy}}$ with a fresh encryption of $\sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}}$, we will replace $\mathsf{ct}_{\mathsf{dummy}}$ with a fresh encryption of $\sigma'||((s_i, \alpha_i, H(t^*))||\ y)$. Here $s_i, \alpha_i, H, y, \sigma'$ are defined as follows: $s_i \leftarrow \mathbb{F}_q^{\rho/\kappa}$ where $q = 2^\kappa$; $H \leftarrow \mathcal{H}$ where $\mathcal{H}$ is a family of collision-resistant hash functions, and $H(t^*)$ is of size κ; $\alpha_i = \langle s_i, t^* \rangle$ is of size κ, where t^* is interpreted as an element in $\mathbb{F}_q^{\rho/\kappa}$, and α_i is interpreted as an element in $\mathbb{F}_q$; $y = \mathsf{PRF.Eval}(k, t^*)$ is of size L_{msg} and $\sigma' = \mathsf{SIG.Sign}(\mathsf{td}, ((s_i, \alpha_i, H(t^*))||\ y))$.

Let sk_i'' denote the secret key after update round i when computed as described above. Let $\mathcal{D}^{\mathcal{A}}_{H_4}$ denote the distribution $\left(\mathsf{pk}', t^*, y, \{f_i(\mathsf{sk}_{i-1}'', \mathsf{sk}_i'')\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$ where $y = C_{\mathsf{Dec}}(\ \mathsf{sk}_n'',\ t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \mathsf{sk}_1'', \ldots, \mathsf{sk}_n''), t^*$ as described above.

Claim 19. *For every* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_{3|n}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_4}.$$

Proof. The proof proceeds through an iteration of sub-hybrid steps, where in the jth iteration we change only the jth ciphertext. Let Hybrid $4^{(j)}$ denote the hybrid game where we have changed only the content of the first j ciphertexts. Let $\mathsf{sk}_i^{(4|j)}$ denote the secret key that is generated in the ith update round of Hybrid $4^{(j)}$. Let $\mathcal{D}^{\mathcal{A}}_{H_{4|j}}$ denote the distribution $\left(\mathsf{pk}', t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(4|j)}, \mathsf{sk}_i^{(4|j)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}')\right)$ where $y = C_{\mathsf{Dec}}(\ \mathsf{sk}_n^{(4|j)},\ t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \mathsf{sk}_0^{(4|j)}, \ldots, \mathsf{sk}_n^{(4|j)}), t^*$

as described above. The proof follows then from the following claim. (Note that Hybrid $4^{(0)}$ is equivalent to Hybrid $3^{(n)}$.) □

Claim 20. *For $j \in \{0, \ldots, n-1\}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_{4|j}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{4|j+1}}$$

Proof. We now define the set of hybrids that allows us to prove Claim 20. We note that, by the definitions of Hybrids $4^{(j)}$ and $4^{(j+1)}$ given above, for any $i \neq j$, $\mathsf{sk}_i^{(4|j)} = \mathsf{sk}_i^{(4|j+1)}$. Therefore, in each of the following sub-hybrids, we only need to make change to $\mathsf{sk}_j^{(4|j)}$.

Hybrid $4a^{(j)}$: In this hybrid, we modify the update procedure in round j. Instead of replacing $\mathsf{ct}_{\mathsf{dummy}}$ with a fresh encryption $\mathsf{RCCA.Enc}(\mathrm{PK}_{\mathsf{RCCA}}, \sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}})$, we set $\mathsf{sk}_j^{(4a|j)} = \mathsf{RCCA.SimEnc}(\tau_{\mathsf{sim}}, \sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}})$. The other keys remain as they are in Hybrid $4^{(j)}$.

Let $\mathsf{sk}_i^{(4a|j)}$ denote the secret key that is generated in the ith update round of Hybrid $4a^{(j)}$. Let $\mathcal{D}^{\mathcal{A}}_{H_{4a|j}}$ denote the distribution $(\mathsf{pk}', t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(4a|j)}, \mathsf{sk}_i^{(4a|j)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}'))$ where $y = C_{\mathsf{Dec}}(\mathsf{sk}_n^{(4a|j)}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}', \mathsf{sk}_0^{(4a|j)}, \ldots, \mathsf{sk}_n^{(4a|j)})$, t^* as described above. We claim the following

Claim 21. *For any* PPT *adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_{4|j}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{4a|j}}$$

Proof. The proof follows by a reduction to the RCCA property ensuring the indistinguishability of simulated ciphertexts from real ciphertexts. Recall, for any efficient adversary $\mathcal{S}$

$$\mathcal{S}^{\mathsf{Enc}(\mathrm{PK}_{\mathsf{RCCA}}, \cdot)}(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_1) \overset{c}{\approx} \mathcal{S}^{\mathsf{SimEnc}(\tau_{\mathsf{sim}}, \cdot)}(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_1) .$$

To build the reduction, $\mathcal{S}$ receives keys $(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_1)$ for the RCCA scheme, which suffices to build the pk of $\mathcal{E}$. For update rounds $i < j$, he constructs sk_i by creating a fresh encryption of $\sigma'||((s_i, \alpha_i, H(t^*))||\ y)$ (as defined above), and for $i > j$, he constructs sk_i by creating a fresh encryption of $\sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}}$. To create sk_j, he submits $\sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}}$ to his challenger and uses the challenge ciphertext in the jth update round. The distribution of secret keys generated by $\mathcal{S}$ is either identical to that in Hybrid $4^{(j)}$, or to that in Hybrid $4a^{(j)}$. It follows that $\mathcal{S}$'s advantage is the same as the distinguishing advantage between $\mathcal{D}^{\mathcal{A}}_{H_{4|j}}$ and $\mathcal{D}^{\mathcal{A}}_{H_{4a|j}}$. □

Hybrid $4b^{(j)}$: In this hybrid, we modify the circuit C_{keys} into $C_{\mathsf{keys}'}$ described in Fig. 20. In words, the change involves using SK_2 instead of SK_1. If decryption under SK_2 outputs a

Internal (hardcoded) state: $\mathsf{keys}' = \{k, \mathsf{vk}, \mathrm{SK}_2\}$.

On input: $\mathsf{ct}_{\mathsf{dummy}}, t$

- If $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, \mathsf{ct}_{\mathsf{dummy}}) = \mathsf{SimFlag}$, output $z = \mathsf{PRF.Eval}(k, t)$.
- If $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, \mathsf{ct}_{\mathsf{dummy}}) = \perp$, output $\perp$.
- Else, parse $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, \mathsf{ct}_{\mathsf{dummy}})$ as (σ', m).
- If $\mathsf{SIG.Verify}(\sigma', m; \mathsf{vk}) = 0$ output $\perp$.
- Otherwise, output $z = \mathsf{PRF.Eval}(k, t)$.

Fig. 20. Program $C_{\mathsf{keys}'}$. This program is obfuscated and placed in the public key, replacing C_{keys}. It is used during decryption.

message, we still verify the signature just as in C_{keys}, and if decryption outputs $\mathsf{SimFlag}$, we proceed as though the signature has been verified. Intuitively, these circuits have differing-inputs security because of the "simulation soundness" property of the RCCA encryption scheme. Denote the resulting public key by $\mathsf{pk}^{(4b|j)}$. Let $\mathcal{D}^{\mathcal{A}}_{H_{4b|j}}$ denote the distribution $\left(\mathsf{pk}^{(4b|j)}, t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(4a|j)}, \mathsf{sk}_i^{(4a|j)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(4b|j)})\right)$ where $y = C_{\mathsf{Dec}}(\mathsf{sk}_n^{(4a|j)}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(4b|j)}, \mathsf{sk}_0^{(4a|j)}, \ldots, \mathsf{sk}_n^{(4a|j)}), t^*$ as described above. We claim the following

Claim 22. *For any efficient adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_{4a|j}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{4b|j}}$$

Proof. We define the following sampler Samp and show that the circuit family $\mathcal{C}$ associated with Samp is a differing-inputs circuit family.
$\mathsf{Samp}(1^\kappa)$ does the following:

- Set $\mathsf{keys} = (\mathrm{SK}_1, k, \mathsf{vk})$ and set $\mathsf{keys}' = (\mathrm{SK}_2, k, \mathsf{vk})$.
- Let $C_0 = C_{\mathsf{keys}}$ and let $C_1 = C_{\mathsf{keys}'}$.
- Set $\mathsf{aux} = (\mathsf{vk}, \{\mathsf{sk}_i^{(4a|j)}\}_{i=0}^n, t^*, y)$
- Return (C_0, C_1, aux).

We now show that for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[C_0(x) \neq C_1(x) : (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Samp}(1^\kappa), x \leftarrow \mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})] \leq \mathsf{negl}(\kappa).$$

Assume toward contradiction that there exists a PPT adversary $\mathcal{A}$ and a polynomial $p(\cdot)$ such that for infinitely many κ, $\mathcal{A}$ outputs a distinguishing input with probability at least $1/p(\kappa)$. We construct a PPT adversary $\mathcal{S}$ that breaks the simulation soundness property of the RCCA scheme.
Upon receiving $(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_1, \mathrm{SK}_2) \leftarrow \mathsf{Gen}(1^\kappa)$ from the challenger, $\mathcal{S}$ does the following:

- Run $k \leftarrow \mathsf{PRF.Gen}(1^\kappa)$ and $(\mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SIG.Gen}(1^\kappa)$. Choose t^* at random, compute $y = \mathsf{PRF.Eval}(k, t^*)$, and set $\mathsf{keys} = (\mathrm{SK}_1, k, \mathsf{vk})$ and $\mathsf{keys}' = (\mathrm{SK}_2, k, \mathsf{vk})$.
- Sample $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{td}, 0^{2\kappa+\rho+L_{\mathsf{msg}}})$ and submit $m^* = (\sigma || 0^{2\kappa+\rho+L_{\mathsf{msg}}})$ as a challenge message.
- $\mathcal{S}$ receives challenge $c^* \leftarrow \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m^*)$ and simulates Samp by doing the following:
  - Let $C_0 = C_{\mathsf{keys}}$ and let $C_1 = C_{\mathsf{keys}'}$.
  - $\mathcal{S}$ sets $\mathsf{sk}_j = c^*$. For $i \neq j$, $\mathcal{S}$ uses $\mathrm{PK}_{\mathsf{RCCA}}$ to generate sk_i honestly, as done in Hybrid $4a^{(j)}$ above.
  - Set $\mathsf{aux} = (\mathsf{vk}, \{\mathsf{sk}_i\}_{i=0}^n, t^*, y)$
- $\mathcal{S}$ runs $\mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})$ and receives x in return. He outputs x.

Note that if $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, x) = \mathsf{SimFlag}$ and $\mathsf{RCCA.Dec}_1(\mathrm{SK}_1, x) = m^*$, then both C_0 and C_1 output $y = \mathsf{PRF.Eval}(k, t^*)$. On the other hand, if $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, x) = \mathsf{SimFlag}$ and $\mathsf{RCCA.Dec}_1(\mathrm{SK}_1, x) \neq m^*$, then x violates the simulation soundness property of the RCCA scheme and $\mathcal{S}$ wins his game. Similarly, if $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, x) \neq \mathsf{SimFlag}$ and $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, x) = \mathsf{RCCA.Dec}_1(\mathrm{SK}_1, x)$, then C_0 and C_1 have the same output (either y or $\perp$). If $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, x) \neq \mathsf{SimFlag}$ and $\mathsf{RCCA.Dec}_2(\mathrm{SK}_2, x) \neq \mathsf{RCCA.Dec}_1(\mathrm{SK}_1, x)$ then, again, $\mathcal{S}$ wins his game.

Claim 22 follows from the fact that diO is a differing-inputs obfuscator and from the fact that the circuit family $\mathcal{C}$ associated with Samp is a differing-inputs family. This is the case since $\mathcal{D}^{\mathcal{A}}_{H_{4a|j}}$ can be simulated given $(\mathsf{diO}(C_0), \mathsf{aux})$ and $\mathcal{D}^{\mathcal{A}}_{H_{4b|j}}$ can be simulated given $(\mathsf{diO}(C_1), \mathsf{aux})$. □

Hybrid $4c^{(j)}$: In this hybrid, we modify the update procedure in round j. Instead of replacing $\mathsf{ct}_{\mathsf{dummy}}$ with the simulated ciphertext $\mathsf{RCCA.SimEnc}(\tau_{\mathsf{sim}}, \sigma || 0^{2\kappa+\rho+L_{\mathsf{msg}}})$, we compute

$$\mathsf{sk}_j^{(4c|j)} = \mathsf{RCCA.SimEnc}(\tau_{\mathsf{sim}}, \sigma' || ((s_i, \alpha_i, H(t^*)) || \ y))$$

as described in Hybrid 4. The other keys remain as they are in Hybrid $4b^{(j)}$.

Let $\mathsf{sk}_i^{(4c|j)}$ denote the secret key that is generated in the ith update round of Hybrid $4c^{(j)}$. Let $\mathcal{D}^{\mathcal{A}}_{H_{4c|j}}$ denote the distribution

$$\left(\mathsf{pk}^{(4b|j)}, t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(4c|j)}, \mathsf{sk}_i^{(4c|j)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(4b|j)})\right)$$

where $y = C_{\mathsf{Dec}}(\mathsf{sk}_n^{(4c|j)}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(4b|j)}, \mathsf{sk}_0^{(4c|j)}, \ldots, \mathsf{sk}_n^{(4c|j)}), t^*$ as described above. We claim the following

Claim 23. *For any efficient adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_{4b|j}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{4c|j}}$$

Proof. The proof follows by a reduction to the RCCA property that ensures the indistinguishability of simulated ciphertexts, even when given SK_2. Recall, for any efficient adversary $\mathcal{S}$, and any message pair (m_0, m_1),

$$\mathcal{A}^{\mathsf{SimEnc}(\tau_{\mathsf{sim}},\cdot)}(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_2, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m_0)) \overset{c}{\approx} \mathcal{A}^{\mathsf{SimEnc}(\tau_{\mathsf{sim}},\cdot)}(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_2, \mathsf{SimEnc}(\tau_{\mathsf{sim}}, m_1)) \ .$$

To build the reduction, $\mathcal{S}$ receives keys $(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_2)$ for the RCCA scheme, which suffices to build the pk of $\mathcal{E}$. For update rounds $i < j$, he constructs sk_i by creating a fresh encryption $\mathsf{RCCA.Enc}(\mathrm{PK}_{\mathsf{RCCA}}, \sigma'||((s_i, \alpha_i, H(t^*))|| \ y))$ (as defined above), and for $i > j$, he constructs sk_i by creating a fresh encryption of $\sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}}$. To create sk_j, he submits challenge plaintext pair: $(\sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}}), (\sigma'||((s_i, \alpha_i, H(t^*))|| \ y))$ to his challenger and uses the challenge ciphertext in the jth update round. The distribution of secret keys generated by $\mathcal{S}$ is either identical to that in Hybrid $4b^{(j)}$, or to that in Hybrid $4c^{(j)}$. It follows that $\mathcal{S}$'s advantage is the same as the distinguishing advantage between $\mathcal{D}^{\mathcal{A}}_{H_{4b|j}}$ and $\mathcal{D}^{\mathcal{A}}_{H_{4c|j}}$. □

Hybrid $4d^{(j)}$ In this hybrid, we modify $C_{\mathsf{keys}'}$ back to the circuit C_{keys} used in the real world. That is, we return to using SK_1. Denote the resulting public key by $\mathsf{pk}^{(4d|j)}$. Let $\mathcal{D}^{\mathcal{A}}_{H_{4d|j}}$ denote the distribution

$$\left(\mathsf{pk}^{(4d|j)}, t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(4c|j)}, \mathsf{sk}_i^{(4c|j)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(4d|j)})\right)$$

where $y = C_{\mathsf{Dec}}(\mathsf{sk}_n^{(4c|j)}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(4d|j)}, \mathsf{sk}_0^{(4c|j)}, \ldots, \mathsf{sk}_n^{(4c|j)})$, t^* as described above. We claim the following

Claim 24. *For any efficient adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_{4c|j}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{4d|j}}$$

Proof. The proof follows from the security of diO. The proof is nearly identical to the proof of Claim 22, so we omit it. □

Hybrid $4e^{(j)}$: In this hybrid, we modify the update procedure in round j. Instead of replacing $\mathsf{ct}_{\mathsf{dummy}}$ with a simulated ciphertext, we return to using a real ciphertext by computing

$$\mathsf{sk}_j^{(4e|j)} = \mathsf{RCCA.Enc}(\sigma'||((s_i, \alpha_i, H(t^*))|| \ y))$$

The other keys remain as they are in hybrid $4d^{(j)}$. Let $\mathsf{sk}_i^{(4e|j)}$ denote the secret key that is generated in the ith update round of hybrid $4e^{(j)}$. Let $\mathcal{D}^{\mathcal{A}}_{H_{4e|j}}$ denote the distribution

$$\left(\mathsf{pk}^{(4d|j)}, t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(4e|j)}, \mathsf{sk}_i^{(4e|j)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(4d|j)})\right)$$

Internal (hardcoded) state: $\mathsf{keys}'' = \{k^* = \mathsf{PRF.Punct}(k, t^*), \mathsf{vk}, \mathrm{SK}_1, H\}$.

On input: $\mathsf{ct}_{\mathsf{dummy}}, t$

- Compute $(\sigma', (s, \alpha, H(t')|| y)) = \mathsf{RCCA.Dec}_1(\mathrm{SK}_1, \mathsf{ct}_{\mathsf{dummy}})$.
- If $\mathsf{SIG.Verify}(\sigma', (s, \alpha, H(t')|| y); \mathsf{vk}) = 0$ output $\perp$.
- If $\langle s, t\rangle = \alpha \bigwedge H(t) = H(t')$, output y.
- Else, output $\mathsf{PRF.Eval}(k^*, t)$.

Fig. 21. Program $C_{\mathsf{keys}''}$. This program replaces C_{keys}. Recall it is obfuscated and placed in the public key. It is used during decryption.

where $y = C_{\mathsf{Dec}}(\mathsf{sk}_n^{(4e|j)}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(4d|j)}, \mathsf{sk}_0^{(4e|j)}, \ldots, \mathsf{sk}_n^{(4e|j)}), t^*$ as described above. We claim the following

Claim 25. *For any efficient adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_{4d|j}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{4e|j}}$$

Proof. The proof is again by the indistinguishability of a simulated ciphertext from a real ciphertext given sk_1. The proof proceeds identically to the proof of Claim 21, so we omit it. □

We note that Hybrid $4e^{(j)}$ is identical to Hybrid $4^{(j+1)}$, so this concludes the proofs of Claims 19 and 20.

Hybrid 5: In this hybrid game, we replace: $C_{\mathsf{Dec}} = \mathsf{diO}(C_{\mathsf{keys}})$ with $C''_{\mathsf{Dec}} = \mathsf{diO}(C_{\mathsf{keys}''})$, where $C_{\mathsf{keys}''}$ is the circuit described in Fig. 21. The difference between keys and keys'' is that we puncture k at the challenge point t^* in keys''. The difference between $C_{\mathsf{keys}''}$ and C_{keys} is that $C_{\mathsf{keys}''}$ will attempt to use the point obfuscation before turning to the PRF key. See Fig. 21 for details. Denote the resulting public key by $\mathsf{pk}^{(5)}$, and let $\mathsf{sk}_i^{(5)}$ denote the secret key after the ith round of update, computed as described in the previous hybrid. Let $\mathcal{D}^{\mathcal{A}}_{H_5}$ denote the distribution $\left(\mathsf{pk}^{(5)}, t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(5)}, \mathsf{sk}_i^{(5)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(5)})\right)$ where $y = C''_{\mathsf{Dec}}(\mathsf{sk}_n^{(5)}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(5)}, \mathsf{sk}_0^{(5)}, \ldots, \mathsf{sk}_n^{(5)}), t^*$ as described above. We claim the following

Claim 26. *For any efficient adversary $\mathcal{A}$,*

$$\mathcal{D}^{\mathcal{A}}_{H_4} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_5}$$

Proof. The proof follows from the security of diO, and from the collision resistance of H and the security of the signature scheme. We define the following sampler Samp and show that the circuit family $\mathcal{C}$ associated with Samp is a differing-inputs circuit family. □

$\mathsf{Samp}(1^\kappa)$ does the following:

- Set $\mathsf{keys} = (\mathrm{SK}_1, k, \mathsf{vk})$ and $\mathsf{keys}'' = (\mathrm{SK}_1, k^*, \mathsf{vk}, H)$.
- Let $C_0 = C_{\mathsf{keys}}$ and let $C_1 = C_{\mathsf{keys}''}$.
- Set $\mathsf{aux} = (\mathsf{vk}, \{\mathsf{sk}_i^{(5)}\}_{i=0}^n, t^*, y)$
- Return (C_0, C_1, aux).

We now show that for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[C_0(x) \neq C_1(x) : (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Samp}(1^\kappa), x \leftarrow \mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})] \leq \mathsf{negl}(\kappa).$$

Assume toward contradiction that there exists a PPT adversary $\mathcal{A}$ and a polynomial $p(\cdot)$ such that for infinitely many κ, $\mathcal{A}$ outputs a distinguishing input with probability at least $1/p(\kappa)$. Denote the event that $\mathcal{A}$ does so by Win. We show that the value output by $\mathcal{A}$ will either enable us to find a collision under H, or to break the existential unforgeability of the signature scheme.

Let $(\mathsf{ct}'_{\mathsf{dummy}}, t')$ denote the output of $\mathcal{A}$ when given circuits and auxiliary input sampled as described above. Let $(\sigma'||m') = \mathsf{RCCA.Dec}_1(\mathsf{sk}_1, \mathsf{ct}'_{\mathsf{dummy}})$. Letting Coll denote the probabilistic event that $(\sigma'||m') = \mathsf{Dec}(\mathrm{SK}_1, \mathsf{sk}_1^{(5)}) = \cdots = \mathsf{Dec}(\mathrm{SK}_1, \mathsf{sk}_n^{(5)})$ (where the randomness is over the coins of the Samp and the coins of $\mathcal{A}$), we divide our analysis into the following two cases.

Claim 27. *There exists an attacker $\mathcal{S}$ that finds collisions on $\mathcal{H}$ with probability* $\Pr[\ \mathsf{Win} \mid \mathsf{Coll}\ ]$.

Proof. Upon receiving $H \leftarrow \mathcal{H}$ from the challenger, $\mathcal{S}$ does the following:

- Run $(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_1) \leftarrow \mathsf{RCCA.Gen}(1^\kappa)$, $k \leftarrow \mathsf{PRF.Gen}(1^\kappa)$ and $(\mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SIG.Gen}(1^\kappa)$. Choose t^* at random, and compute $k^* = \mathsf{PRF.Punct}(k, t^*)$. Set $\mathsf{keys} = (\mathrm{SK}_1, k, \mathsf{vk})$ and $\mathsf{keys}'' = (\mathrm{SK}_1, k^*, \mathsf{vk}, H)$.
- $\mathcal{S}$ simulates Samp by doing the following:
  - $\mathcal{S}$ samples $s \leftarrow \mathbb{F}_q^{\rho/\kappa}$ and computes $\alpha = \langle s, t^* \rangle$. He computes $y = \mathsf{PRF.Eval}(k, t^*)$. He uses these values, along with challenge H and signing key td, to generate $\mathsf{sk}_i^{(5)}$ honestly.
  - Set $\mathsf{aux} = (\mathsf{vk}, \{\mathsf{sk}_i^{(5)}\}_{i=0}^n, t^*, y)$
- $\mathcal{S}$ runs $\mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})$ and receives $x = (\mathsf{ct}''_{\mathsf{dummy}}, t'')$ in return. He outputs t'' as a collision with t^* under H.

Because we condition on event Coll, we have that $\mathsf{RCCA.Dec}_1(\mathrm{SK}_1, \mathsf{ct}''_{\mathsf{dummy}}) = \sigma''||(s, \alpha, H(t^*)||\ y)$, where s and t^* are the values sampled by $\mathcal{S}$ when simulating $\mathsf{sk}_i^{(5)}$, $\alpha = \langle s, t^* \rangle$, and $\mathsf{SIG.Verify}(\sigma'', (s, \alpha, H(t^*)||\ y)) = 1$. It follows that the only way for $(\mathsf{ct}''_{\mathsf{dummy}}, t'')$ to constitute a differing-inputs is if the following condition holds:

$$t'' \neq t^* \bigwedge \mathsf{PRF.Eval}(k, t'') \neq y \bigwedge \langle s, t'' \rangle = \alpha \bigwedge H(t'') = H(t^*)$$

To see why this condition is necessary, note that if $t'' = t^*$, or if $\mathsf{PRF.Eval}(k, t'') = y$, both circuits output y. If $\langle s, t'' \rangle \neq \alpha$, or $H(t'') \neq H(t^*)$, both circuits output $\mathsf{PRF.Eval}(k, t'')$. Now, since $t'' \neq t^*$, but $H(t'') = H(t^*)$, $\mathcal{S}$ has succeeded in finding a collision for function H. □

Claim 28. *There exists an attacker $\mathcal{S}$ that finds forgeries with respect to* Sign *with probability*
$\Pr[\ \mathsf{Win} \mid \overline{\mathsf{Coll}}\ ]$.

Proof. Upon receiving vk from the challenger, $\mathcal{S}$, who has the access to the signing oracle $\mathsf{SIG.Sign}(\mathsf{td}, \cdot)$, does the following:

- Run $(\mathrm{PK}_{\mathsf{RCCA}}, \mathrm{SK}_1) \leftarrow \mathsf{RCCA.Gen}(1^\kappa)$, $k \leftarrow \mathsf{PRF.Gen}(1^\kappa)$. Choose t^* at random, and compute $k^* = \mathsf{PRF.Punct}(k, t^*)$. Choose $H \leftarrow \mathcal{H}$. Set $\mathsf{keys} = (\mathrm{SK}_1, k, \mathsf{vk})$ and $\mathsf{keys}'' = (\mathrm{SK}_1, k^*, \mathsf{vk}, H)$.
- $\mathcal{S}$ simulates Samp by doing the following:
  - $\mathcal{S}$ samples $s \leftarrow \mathbb{F}_q^{\rho/\kappa}$ and computes $\alpha = \langle s, t^* \rangle$. He computes $y = \mathsf{PRF.Eval}(k, t^*)$. He uses these values to define message $m = (s, \alpha, H(t^*)||y)$ and uses his external signing oracle $\mathsf{SIG.Sign}(\mathsf{td}, \cdot)$ to get signature σ. He generates $\mathsf{sk}_i^{(5)}$ by repeatedly encrypting (σ, m).
  - Set $\mathsf{aux} = (\mathsf{vk}, \{\mathsf{sk}_i^{(5)}\}_{i=0}^n, t^*, y)$
- $\mathcal{S}$ runs $\mathcal{A}(1^\kappa, C_0, C_1, \mathsf{aux})$ and receives $x = (\mathsf{ct}''_{\mathsf{dummy}}, t'')$ in return. Then he computes $(\sigma'', m'') = \mathsf{RCCA.Dec}_1(\mathrm{SK}_1, \mathsf{ct}''_{\mathsf{dummy}})$, and outputs (m'', σ'') as a forged signature.

Because we are conditioning on the event $\overline{\mathsf{Coll}}$, it follows that $\mathcal{S}$ never obtains (σ'', m'') through his signing oracle. Furthermore, note that if $\mathsf{SIG.Verify}(\mathsf{vk}, m'', \sigma'') = 0$, then both C_0 and C_1 output $\perp$. It follows that (m'', σ'') is a successful forgery, and $\mathcal{S}$ wins his game. Note that we require a *strongly* unforgeable signature scheme, because, while $\overline{\mathsf{Coll}}$ states that $(m'', \sigma'') \neq (m, \sigma)$, it may be that $m'' = m$. □

From Claims 27 and 28, and the security of $\mathcal{H}$ and Sign, it follows that $\Pr[\mathsf{Win}] < \mathsf{negl}$. Claim 24 follows from the fact that diO is a differing-inputs obfuscator and from the fact that the circuit family $\mathcal{C}$ associated with Samp is a differing-inputs family. This is the case since $\mathcal{D}^{\mathcal{A}}_{H_4}$ can be simulated given $(\mathsf{diO}(C_0), \mathsf{aux})$ and $\mathcal{D}^{\mathcal{A}}_{H_5}$ can be simulated given $(\mathsf{diO}(C_1), \mathsf{aux})$.

Hybrid 6: In this game, we modify the key update phase (in every round) to use $(s, \alpha, H(t^*)||\ y^*)$, where y^* is chosen uniformly at random, rather than as $\mathsf{PRF.Eval}(k, t^*)$. Let $\mathcal{D}^{\mathcal{A}}_{H_6}$ denote the distribution
$\left(\mathsf{pk}^{(6)}, t^*, y, \{f_i(\mathsf{sk}_{i-1}^{(6)}, \mathsf{sk}_i^{(6)})\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(6)})\right)$ where $y = C''_{\mathsf{Dec}}(\ \mathsf{sk}_n^{(6)}, t^*)$ and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(6)}, \mathsf{sk}_0^{(6)}, \ldots, \mathsf{sk}_n^{(6)}), t^*$ as described above. We claim the following

Claim 29. *For any* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_5} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_6}$$

Proof. The proof is by reduction to the security of the punctured PRF. Specifically, $\mathcal{S}$ attacks the PRF by submitting t^* to his challenger and receiving $(\mathsf{PRF.Punct}(k, t^*), y^*)$ as a challenge. He then generates all the other necessary keys to simulate the view of $\mathcal{A}$. And uses $\mathcal{A}$'s guess to form his own. □

Hybrid 7: In this game, we replace C''_{Dec} with C'''_{Dec} by changing k^* in C''_{Dec} in the previous hybrid with the original k.

Claim 30. *For any* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_6} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_7}$$

Proof. The proof is by a reduction to the security of the indistinguishability obfuscation. The main observation is that if $H(t') = H(t)$ and $\langle s, t\rangle = \alpha$, then both circuits output y^*. If this does not hold, then C''_{Dec} returns $y'' = \mathsf{PRF.Eval}(k^*, t)$, and C'''_{Dec} returns $y''' = \mathsf{PRF.Eval}(k, t)$; note that $y'' = y'''$ on points $t \neq t*$. Therefore, changing k^* into k will not effect the input/output behavior. If there is a difference in advantage, we can create an algorithm $\mathcal{B}$ that breaks the security of indistinguishability obfuscation. □

Hybrid $8^{(j)}$: In this sequence of hybrids, we modify the key update procedure as follows. Instead of replacing $\mathsf{ct}_{\mathsf{dummy}}$ with an encryption of $(s, \alpha, H(t^*)||\ y^*)$, where $\langle s, t^*\rangle = \alpha$, in the first j key update rounds we instead replace it with a fresh encryption of $(s_i, \alpha_i, H(t^*)||\ y^*)$ where $s_i \leftarrow \mathbb{F}_q^{\rho/\kappa}$ and $\alpha_i \leftarrow \mathbb{F}_q$. Note that the difference in this hybrid is that α_i is no longer necessarily equal to $\alpha = \langle s_i, t^*\rangle$. Let $\mathcal{D}^{\mathcal{A}}_{H_{8|j}}$ denote the distribution $\left(\mathsf{pk}^{(8|j)}, t^*, y^*, \{f_i(\mathsf{sk}^{(8|j)}_{i-1}, \mathsf{sk}^{(8|j)}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(8|j)})\right)$ where $y^* \leftarrow \{0,1\}^{L_{\mathsf{msg}}}$ is chosen uniformly at random, and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(8|j)}, \mathsf{sk}^{(8|j)}_0, \ldots, \mathsf{sk}^{(8|j)}_n)$, w, and y^* as described above. Noting that Hybrid $8^{(0)}$ is the same as Hybrid 7, We claim the following

Claim 31. *For any* PPT *adversary* $\mathcal{A}$*, and any* $j \in \{0, \ldots, n-1\}$

$$\mathcal{D}^{\mathcal{A}}_{H_{8|j}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{8|j+1}}$$

The heart of this proof relies on Theorem 15 which is stated and proved next. Intuitively, the claim in this theorem is that, for any leakage function f with bounded output length, it is hard to tell from $(t^*, f(s_j, H(t^*), \alpha_j, \alpha_{j+1}))$ whether $\alpha_j = \langle s_j, t^*\rangle$. The argument uses the fact that the inner product is a strong two-source extractor.

Lemma 17. *(Strong Inner-Product Two-Source Extractor) Let* $\boldsymbol{X}, \boldsymbol{Y}, Z$ *be correlated variables, where* $\boldsymbol{X}, \boldsymbol{Y}$ *have their support in* $\mathbb{F}_q^m$*, and are independent conditioned on* Z*.*

Let U be uniform and independent on $\mathbb{F}_q$. Then

$$\Delta((Z, Y, \langle \boldsymbol{X}, \boldsymbol{Y} \rangle), (Z, Y, U)) \leq 2^{-s}$$

for some $s \geq 1 + \frac{1}{2}(k_X + k_Y - (m+1)\log q)$, where $k_X := \widetilde{\mathbf{H}}_\infty(\boldsymbol{X}|Z)$, $k_Y := \widetilde{\mathbf{H}}_\infty(\boldsymbol{Y}|Z)$

The worst-case version of this lemma is Theorem 1 of Lee et al. [42]. The average-case version that we use above follows as in Wichs' thesis [51, Lemma 4.1.4]. (Wichs does not state his lemma for the *strong* extraction property but this follows readily given the result of Lee et al. [42].)

Theorem 15. *Let S_1, T be random on $\mathbb{F}_q^m$ and U be random on $\mathbb{F}_q$. Fix any $s_2 \in \mathbb{F}_q^m$ and let $A_2 = \langle s_2, T \rangle$. Suppose H outputs κ bits and f outputs L' bits. Then*

$$\Delta\left((T, f(S_1, H(T), \langle S_1, T\rangle, A_2)), (T, f(S_1, H(T), U, A_2))\right) \leq 2^{-s'}, \tag{6}$$

where $s' = (m \log q - 3L' - 1 - \kappa - 2\log q)/3$.

Thus for the statistical distance in Eq. 6 to be negligible, we need

$$(m \log q - 3L' - 1 - \kappa - 2\log q)/3 \geq \log(1/\epsilon') \ ,$$

for some negligible ϵ'. Taking $\kappa = \log 1/\epsilon' = \log q$ and setting $m = \omega(\kappa)$ such that $m \cdot \kappa = \rho(\kappa)$, $L' = \rho/3 - O(\kappa)$, we can tolerate 2CLR leakage functions of length $L = L'/2$, which we will argue later.

Proof. Let BAD be the set of ℓ such that conditioning on $f(S_1, H(T), U, A_2) = \ell$ gives S_1 too little min-entropy. That is,

$$\mathsf{BAD} := \{\ell \text{ such that } \mathbf{H}_\infty(S_1 \mid f(S_1, H(T), U, A_2) = \ell) < m \log q - L' - s' - 1\} \ .$$

Additionally, for fixed t, $\alpha_2 = \langle s_2, t \rangle$, let $\mathcal{S}^t$ be the set of ℓ defined as,

$$\mathcal{S}^t := \{\ell \text{ such that } \Pr[f(S_1, H(t), U, \alpha_2) = \ell] > \Pr[f(S_1, H(t), \langle S_1, t\rangle, \alpha_2) = \ell]\}.$$

□

We claim that

$$\begin{aligned}
&\Delta\left((T, f(S_1, H(T), U, A_2)), (T, f(S_1, H(T), \langle S_1, T\rangle, A_2))\right) \\
&= \mathbf{E}_t \ \Delta\left(f(S_1, H(t), U, \alpha_2), f(S_1, H(t), \langle S_1, t\rangle, \alpha_2)\right) \\
&\leq \mathbf{E}_t \left(\sum_{\ell \in \overline{\mathsf{BAD}} \cap \mathcal{S}^t} \Pr\left[f(S_1, H(t), U, \alpha_2) = \ell\right] - \Pr\left[f(S_1, H(t), \langle S_1, t\rangle, A_2) = \ell\right] \right) \\
&+ \mathbf{E}_t \left(\sum_{\ell \in \mathsf{BAD} \cap \mathcal{S}^t} \Pr\left[f(S_1, H(t), U, \alpha_2) = \ell\right] \right)
\end{aligned}$$

$$
\begin{aligned}
&\leq \sum_{\ell \notin \mathsf{BAD}} \left(\mathbf{E}_t \left|\Pr\left[f(S_1, H(t), U, \alpha_2) = \ell\right] - \Pr\left[f(S_1, H(t), \langle S_1, t\rangle, \alpha_2) = \ell\right]\right|\right) \\
&+ \sum_{\ell \in \mathsf{BAD}} \left(\mathbf{E}_t \Pr\left[f(S_1, H(t), U, \alpha_2) = \ell\right]\right) \\
&= \sum_{\ell \notin \mathsf{BAD}} \left(\mathbf{E}_t \left|\Pr\left[f(S_1, H(t), U, \alpha_2) = \ell\right] - \Pr\left[f(S_1, H(t), \langle S_1, t\rangle, \alpha_2) = \ell\right]\right|\right) \\
&+ \Pr[f(S_1, H(T), U, A_2) \in \mathsf{BAD}] \\
&\leq \sum_{\ell \notin \mathsf{BAD}} (\mathbf{E}_t |\Pr[f(S_1, H(t), U, \alpha_2) = \ell] \\
&- \Pr[f(S_1, H(t), \langle S_1, t\rangle, \alpha_2) = \ell]|) + 2^{-s'-1} \qquad (7) \\
&\leq 2^{L'} \cdot 2^{-s'-1-L'} + 2^{-s'-1}, \\
&= 2^{-s'}. \qquad (8)
\end{aligned}
$$

where (7) is due to Markov's inequality and the definition of the set BAD. (8) is due to the following claim:

Claim 32. *For any $\ell \notin \mathsf{BAD}$ in the range of f,*

$$\mathbf{E}_t \left|\Pr\left[f(S_1, H(t), U, \alpha_2) = \ell\right] - \Pr\left[f(S_1, H(t), \langle S_1, t\rangle, \alpha_2) = \ell\right]\right| \leq 2^{-s'-1-L'}.$$

Proof. First, for fixed t, β and $\widehat{h} = H(t)$, we have:

$$
\begin{aligned}
&\Pr[f(S_1, \widehat{h}, \langle S_1, t\rangle, \alpha_2) = \ell \wedge \langle S_1, t\rangle = \beta] \\
&\quad = \Pr[f(S_1, \widehat{h}, \beta, \alpha_2) = \ell \wedge \langle S_1, t\rangle = \beta] \\
&\quad = \Pr[f(S_1, \widehat{h}, \beta, \alpha_2) = \ell] \cdot \Pr[\langle S_1, t\rangle = \beta \mid f(S_1, \widehat{h}, \beta, \alpha_2) = \ell]. \qquad (9)
\end{aligned}
$$

and

$$
\begin{aligned}
&\Pr[f(S_1, \widehat{h}, U, \alpha_2) = \ell \wedge U = \beta] \\
&\quad = \Pr[f(S_1, \widehat{h}, \beta, \alpha_2) = \ell \wedge U = \beta] \\
&\quad = \Pr[f(S_1, \widehat{h}, \beta, \alpha_2) = \ell] \cdot \Pr[U = \beta]. \qquad (10)
\end{aligned}
$$

Now, we have that:

$$
\begin{aligned}
&\mathbf{E}_{t \leftarrow T} |\Pr[f(S_1, H(t), U, \alpha_2) = \ell] - \Pr[f(S_1, H(t), \langle S_1, t\rangle, \alpha_2) = \ell]| \\
&\quad = \sum_{\widehat{h}, \alpha_2} \Pr[H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2] \cdot \mathop{\mathbf{E}}_{t \leftarrow T'} |\Pr[f(S_1, \widehat{h}, U, \alpha_2) = \ell] \\
&\quad - \Pr[f(S_1, \widehat{h}, \langle S_1, t\rangle, \alpha_2) = \ell]| \\
&\quad \leq \sum_{\widehat{h}, \alpha_2} \Pr[H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2]
\end{aligned}
$$

$$
\begin{aligned}
&\cdot \mathop{\mathbf{E}}_{t \leftarrow T'} \sum_{\beta} |\Pr[f(S_1, \widehat{h}, U, \alpha_2) = \ell \wedge U = \beta] \\
&- \Pr[f(S_1, \widehat{h}, \langle S_1, t\rangle, \alpha_2) = \ell \wedge \langle S_1, t\rangle = \beta]| \qquad (11)\\
&= \sum_{\widehat{h}, \alpha_2} \Pr[H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2] \\
&\cdot \sum_{\beta} (\mathop{\mathbf{E}}_{t \leftarrow T'} |\Pr[f(S_1, \widehat{h}, U, \alpha_2) = \ell \wedge U = \beta] \\
&- \Pr[f(S_1, \widehat{h}, \langle S_1, t\rangle, \alpha_2) = \ell \wedge \langle S_1, t\rangle = \beta]|) \\
&= \sum_{\widehat{h}, \alpha_2} \Pr[H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2] \\
&\cdot \sum_{\beta} (\mathop{\mathbf{E}}_{t \leftarrow T'} [\Pr[f(S_1, \widehat{h}, \beta, \alpha_2) = \ell] \cdot |\Pr[U = \beta] \\
&- \Pr[\langle S_1, t\rangle = \beta \mid f(S_1, \widehat{h}, \beta, \alpha_2) = \ell]|]) \qquad (12)\\
&\le \sum_{\widehat{h}, \alpha_2} \Pr[H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2] \cdot \sum_{\beta} (\mathop{\mathbf{E}}_{t \leftarrow T'} |\Pr[U = \beta] \\
&- \Pr[\langle S_1, t\rangle = \beta \mid f(S_1, \widehat{h}, \beta, \alpha_2) = \ell]|) \qquad (13)\\
&= \sum_{\widehat{h}, \alpha_2} \Pr[H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2] \\
&\Delta((T', (\langle S_1, T'\rangle \mid f(S_1, \widehat{h}, \beta, \alpha_2) = \ell)), (T', U)) \\
&\le 2^{-s'-1-L'}, \qquad (14)
\end{aligned}
$$

where T' is uniform on the set $\{t \mid H(t) = \widehat{h} \wedge \langle s_2, t\rangle = \alpha_2\}$, (11) follows by triangle inequality, (12) follows from (9) and (10), (13) follows since the quantity $\Pr[f(S_1, \widehat{h}, \beta, \alpha_2) = \ell]$ is always less than or equal to 1.

To see why (14) holds, note that $X := \left(S_1 \mid f(S_1, \widehat{h}, \beta, \alpha_2) = \ell\right)$ and $Y := \left(T \mid H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2\right)$ are independent sources. Furthermore,

$$k_X := \widetilde{\mathbf{H}}_{\infty}\left(S_1 \mid f(S_1, \widehat{h}, \beta, \alpha_2) = \ell\right) \ge m \log q - L' - s' - 1$$

by the assumption $\ell \notin \mathsf{BAD}$. And

$$k_Y := \widetilde{\mathbf{H}}_{\infty}\left(T \mid H(T) = \widehat{h} \wedge \langle s_2, T\rangle = \alpha_2\right) \ge m \log q - \kappa - \log q$$

by the "chain rule" for average min-entropy [25, Lemma 2.2]. Thus the last inequality follows by Lemma 17. □

We now use theorem 15 to prove Claim 31.

Proof. We show that if $\mathcal{A}$ can distinguish Hybrid $8^{(j)}$ from $8^{(j+1)}$ with some advantage ϵ', then there is a distinguisher $\mathcal{S}$ and functions f^* and H with output lengths at most

Internal (hardcoded) state: $\mathsf{rand} = \left\{\{s_i, \alpha_i, r_i^{(1)}, r_i^{(2)}\}_{i=0}^{j-1}, r_j^{(1)}, r_j^{(2)}, r_{j+1}^{(1)}, r_{j+1}^{(2)}, y^*, r_{\mathsf{tape}}\right\}$, td, $\mathsf{pk}_{\mathsf{RCCA}}$.

On input: $s_j, H(t^*), \alpha_j, \alpha_{j+1}$

- For $i \in \{0, \ldots, j+1\}$,
  - let $m_i = (s_i, H(t^*), \alpha_i || y^*)$,
  - let $\sigma_i = \mathsf{SIG.Sign}(\mathsf{td}, m_i\,; r_i^{(1)})$,
  - let $\mathsf{sk}_i = \mathsf{RCCA.Enc}(\mathsf{pk}_{\mathsf{RCCA}}, \sigma_i \,||\, m_i\,; r_i^{(2)})$.
- Run the code of $\mathcal{A}$ using random tape r_{rand} until he has made $j+1$ leakage queries. For $i \in \{1, \ldots, j+1\}$, reply to leakage query f_i by computing $f_i(\mathsf{sk}_{i-1}, \mathsf{sk}_i)$.
- Output $f_j(\mathsf{sk}_{j-1}, \mathsf{sk}_j), f_{j+1}(\mathsf{sk}_j, \mathsf{sk}_{j+1})$

Fig. 22. The function $f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}$ is the leakage function sent by reduction adversary $\mathcal{S}$ to his challenger. In response, he receives either a sample from $(T, f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}(S_1, H(T), \langle S_1, T\rangle, \alpha_2))$, or from $(T, f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}(S_1, H(T), \beta, \alpha_2))$.

$L' = 2L$ bits and κ bits, respectively, such that $\mathcal{S}$ distinguishes the two distributions $(T, f^*(S_1, H(T), \langle S_1, T\rangle, A_2))$ and $(T, f^*(S_1, H(T), U, A_2))$ from one another with the same advantage. Since this violates the assertion in Theorem 15, the claim follows directly.

$\mathcal{S}$ simulates the view of $\mathcal{A}$ as follows. He samples $(\mathsf{pk}_{\mathsf{RCCA}}, \mathsf{sk}_1, \mathsf{sk}_2) \leftarrow \mathsf{RCCA.Gen}(1^\kappa)$, $(\mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SIG.Gen}(1^\kappa)$, $k \leftarrow \mathsf{PRF.Gen}(1^\kappa)$, and constructs C_{Dec} and C_{Enc} as described in the previous hybrid. Then, to simulate the replies to $\mathcal{A}$'s leakage queries, $\mathcal{S}$ acts as follows. For $i \in \{0, \ldots, j-1\}$, he samples $s_i \leftarrow \mathbb{F}_q^{\rho/\kappa}$, and $\alpha_i \leftarrow \mathbb{F}_q$. For $i \in \{0, \ldots, j+1\}$ he samples randomness $r_i^{(1)}$ to be used in signing the necessary plaintext values, and $r_i^{(2)}$ to be used in encrypting the necessary values. Finally, he samples r_{tape} to be used as $\mathcal{A}$'s random tape, and $y^* \leftarrow \{0,1\}^{L_{\mathsf{msg}}}$. We denote the union of these sets of random values by rand. He then submits function $f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}$ to his challenger, where $f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}$ is defined as in Fig. 22.

$\mathcal{S}$ receives challenge value $(t^*, f_{j-1}(\mathsf{sk}_{j-2}, \mathsf{sk}_{j-1}), f_j(\mathsf{sk}_{j-1}, \mathsf{sk}_j), \alpha_{j+1})$. Once he knows t^*, we note that $\mathcal{S}$ has all the information needed to simulate $\mathcal{A}$'s view for the first $j-2$ leakage queries. He does so precisely as was done by his challenger when running $f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}$, using identical random values, and eliciting identical leakage queries. To simulate the replies to leakage queries f_{j-1} and f_j he uses the two outputs of $f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}$.

By our setting of parameters, output size of f is L, and output size of f^* is $L' = 2L$. To simulate the replies to query f_{j+1}, $\mathcal{S}$ computes $\alpha_{j+1} = \langle s_{j+1}, t^*\rangle$, where, recall, s_{j+1} was fixed prior to his challenge query. He simulates $\mathsf{sk}_{j+1}^{(8|j)}$ by signing and encrypting $(s_{j+1}, H(t^*), \alpha_{j+1}||y^*)$ with random coins $r_{j+1}^{(1)}, r_{j+1}^{(2)}$. For $j+1 < i < n$, he constructs $\mathsf{sk}_i^{(8|j)}$ by sampling $s_i \leftarrow \mathbb{F}_q^{\rho/\kappa}$, computing $\alpha_i = \langle s_i, t^*\rangle$, and then signing and encrypting $(s_i, H(t^*), \alpha_i||y^*)$ using uniformly chosen coins. He simulates leakage queries $f_{j+1}, \ldots, f_n$ using the $\mathsf{sk}_{j+1}^{(8|j)}, \ldots, \mathsf{sk}_n^{(8|j)}$.

The above simulation is distributed exactly as Hybrid $8^{(j)}$ when $\mathcal{S}$'s challenge comes from $(T, f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}(S_j, H(T), \langle S_j, T\rangle, \alpha_{j+1}))$, and it is distributed exactly as Hybrid $8^{(j+1)}$ when $\mathcal{S}$'s challenge comes from $(T, f^*_{\mathsf{rand},\mathsf{td},\mathsf{pk}_{\mathsf{RCCA}}}(S_j, H(T), \beta, \alpha_{j+1}))$, which concludes the proof. □

Hybrid 9: In this hybrid, we change the circuit C_{Dec} such that after decrypting $\mathsf{ct}_{\mathsf{dummy}}$, it verifies the signature, but otherwise ignores the content. Note that by the end of Hybrid 8 sequence, the inner-product relationship has been "broken", so with all but negligible probability over the choices of s_i, α_i, we are already ignoring the plaintext values anyway.

Let $C^{(9)}_{\mathsf{keys}}$ denote the resulting circuit, and let $\mathsf{pk}^{(9)}$ denote the modified public key that results from obfuscating the updated circuit. Let $\mathcal{D}^{\mathcal{A}}_{H_9}$ denote the distribution $\left(\mathsf{pk}^{(9)}, t^*, y^*, \{f_i(\mathsf{sk}^{(9)}_{i-1}, \mathsf{sk}^{(9)}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(9)})\right)$ where y^* is chosen uniformly at random, and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(8)}, \mathsf{sk}^{(8)}_0, \ldots, \mathsf{sk}^{(8)}_n)$, y^* as described above. We claim the following

Claim 33. *For any* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_{8|n}} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_9}$$

Proof. The proof follows from a reduction to the security of the diO scheme. The argument that these two circuits have differing-inputs security follows almost identically as in the proof of Claim 26, with a reduction to the either the security of the signature scheme, or the collision resistance of H. We omit repeating the proof. □

Hybrid 10: In this hybrid, we replace the content of $\mathsf{ct}_{\mathsf{dummy}}$ with a fresh encryption of $\sigma||0^{2\kappa+\rho+L_{\mathsf{msg}}}$, where σ is a signature on $0^{2\kappa+\rho+L_{\mathsf{msg}}}$. Let $\mathsf{sk}^{(10)}_i$ denote the resulting secret key in update round i. Let $\mathsf{pk}^{(10)}$ denote the public key (which is generated in the same fashion as in Hybrid 9.). Let $\mathcal{D}^{\mathcal{A}}_{H_{10}}$ denote the distribution $(\mathsf{pk}^{(10)}, t^*, y^*, \{f_i(\mathsf{sk}^{(10)}_{i-1}, \mathsf{sk}^{(10)}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk}^{(10)}))$ where y^* is chosen uniformly at random, and the distribution is taken over coins of $\mathcal{A}$, and choice of $(\mathsf{pk}^{(10)}, \mathsf{sk}^{(10)}_0, \ldots, \mathsf{sk}^{(10)}_n)$, y^* as described above. We claim the following

Claim 34. *For any* PPT *adversary* $\mathcal{A}$*,*

$$\mathcal{D}^{\mathcal{A}}_{H_9} \overset{c}{\approx} \mathcal{D}^{\mathcal{A}}_{H_{10}}$$

Proof. The proof follows from the security of the RCCA scheme. We transition through a sequence of hybrids (and sub-hybrids), changing one plaintext value at a time, just as we did in Claim 20. Note that, just as in that claim, our circuit only verifies the signature on the plaintext, and makes no use of the value otherwise. Since we only need to verify the signature, we can replace decryption with SK_1 by a check for $\mathsf{SimFlag}$ after decrypting with SK_2, and use diO security, as we did previously. We omit the details of the proof.

Finally, we have the following claim, where the right-hand side is the same as in Lemma 16.

Claim 35.

$$\mathcal{D}^{\mathcal{A}}_{H_{10}} \overset{s}{\approx} \left(\widetilde{\mathsf{pk}}, U_\rho, U_{L_{\mathsf{msg}}}, \{f_i(\widetilde{\mathsf{sk}}_{i-1}, \widetilde{\mathsf{sk}}_i)\}_{i=1}^n \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\widetilde{\mathsf{pk}})\right)$$

Proof. Note that $\mathsf{pk}^{(10)}$ and $\mathsf{sk}_0^{(10)}, \ldots, \mathsf{sk}_n^{(10)}$ contain no information about y^*. □

This concludes the proof of Lemma 16. □

References

[1] M. Abe, M. Chase, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, vol. 7658 of *LNCS* (Springer, Berlin, 2012), pp. 4–24.

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In O. Reingold, editor, *TCC 2009*, vol. 5444 of *LNCS*. (Springer, Berlin, 2009), pp. 474–495.

[3] P. Ananth, D. Boneh, S. Garg, A. Sahai, M. Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. http://eprint.iacr.org/2013/689.

[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *CRYPTO 2001*, vol. 2139 of *LNCS*. (Springer, Berlin, 2001), pp. 1–18.

[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[6] A. Boldyreva, S. Fehr, and A. O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In D. Wagner, editor, *CRYPTO 2008*, vol. 5157 of *LNCS*. (Springer, Berlin, 2008), pp. 335–359.

[7] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS* (Springer, Berlin, 2004), pp. 41–55.

[8] D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, vol. 8270 of *LNCS* (Springer, Berlin, 2013), pp. 280–300.

[9] E. Boyle, K.-M. Chung, R. Pass. On extractability obfuscation. In Y. Lindell, editor, *TCC 2014*, vol. 8349 of *LNCS* (Springer, Berlin, 2014), pp. 52–73.

[10] E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In H. Krawczyk, editor, *PKC 2014*, vol. 8383 of *LNCS* (Springer, Berlin, 2014), pp. 501–519.

[11] E. Boyle, G. Segev, and D. Wichs. Fully leakage-resilient signatures. In K. G. Paterson, editor, *EUROCRYPT 2011*, vol. 6632 of *LNCS* (Springer, Berlin, 2011), pp. 89–108.

[12] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51st FOCS*, pp. 501–510. IEEE Computer Society Press, (2010).

[13] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In B. S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*. (Springer, Berlin, 1997), pp. 90–104.

[14] R. Canetti, S. Goldwasser, and O. Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part II*, vol. 9015 of *LNCS* (Springer, Berlin, 2015), pp. 557–585.

[15] R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing chosen-ciphertext security. In D. Boneh, editor, *CRYPTO 2003*, vol. 2729 of *LNCS* (Springer, Berlin, 2003), pp. 565–582.

[16] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *CRYPTO'99*, vol. 1666 of *LNCS*. (Springer, Berlin, 1999)

[17] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, vol. 7237 of *LNCS* (Springer, Berlin, 2012), pp. 281–300

[18] D. Dachman-Soled, J. Katz, and V. Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part II*, vol. 9015 of *LNCS* (Springer, Berlin, 2015), pp. 586–613

[19] D. Dachman-Soled, F.-H. Liu, and H.-S. Zhou. Leakage-resilient circuits revisited - optimal number of computing components without leak-free hardware. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, vol. 9057 of *LNCS*, (Springer, Berlin, 2015), pp. 131–158.

[20] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In J. Kilian, editor, *CRYPTO 2001*, vol. 2139 of *LNCS* (Springer, Berlin, 2001), pp. 566–598

[21] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *51st FOCS*, IEEE Computer Society Press, 2010, pp. 511–520.

[22] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Efficient public-key cryptography in the presence of key leakage. In M. Abe, editor, *ASIACRYPT 2010*, vol. 6477 of *LNCS* (Springer, Berlin, 2010), pp. 613–631.

[23] Y. Dodis, Y. T. Kalai, and S. Lovett. On cryptography with auxiliary input. In M. Mitzenmacher, editor, *41st ACM STOC* (ACM Press, 2009), pp. 621–630.

[24] Y. Dodis, A. B. Lewko, B. Waters, and D. Wichs. Storing secrets on continually leaky devices. In R. Ostrovsky, editor, *52nd FOCS*, pp. 688–697. IEEE Computer Society Press, 2011.

[25] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[26] Y. Dodis and A. Smith. Correcting errors without leaking partial information. In H. N. Gabow and R. Fagin, editors, *37th ACM STOC* (ACM Press, 2005), pp. 654–663.

[27] S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In H. Gilbert, editor, *EUROCRYPT 2010*, vol. 6110 of *LNCS* (Springer, Berlin, 2010), pp. 135–156.

[28] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, vol. 7881 of *LNCS* (Springer, Berlin, 2013), pp. 1–17.

[29] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. in *54th FOCS*, pp. 40–49. IEEE Computer Society Press, 2013.

[30] S. Garg, C. Gentry, S. Halevi, and D. Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, (Springer, Berlin, 2014), pp. 518–535.

[31] S. Garg and A. Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part II*, vol. 9015 of *LNCS* (Springer, Berlin, 2015), pp. 614–637.

[32] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, Aug. 1986.

[33] S. Goldwasser and G. N. Rothblum. On best-possible obfuscation. In S. P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS* (Springer, Berlin 2007), pp. 194–213

[34] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys, in *USENIX Security Symposium*, pp. 45–60 (2008)

[35] C. Hazay, A. López-Alt, H. Wee, and D. Wichs. Leakage-resilient cryptography from minimal assumptions. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, (Springer, Berlin, 2013), pp. 160–176.

[36] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC* (ACM Press, 1989), pp. 12–24.

[37] Y. Ishai, O. Pandey, and A. Sahai. Public-coin differing-inputs obfuscation and its applications. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part II*, vol. 9015 of *LNCS*, (Springer, Berlin, 2015), pp. 668–697.

[38] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[39] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In M. Matsui, editor, *ASIACRYPT 2009*, vol. 5912 of *LNCS*, (Springer, Berlin, 2009), pp. 703–720

[40] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. Massachusetts Institute of Technology (1994)

[41] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, (ACM Press, 2013), pp. 669–684.

[42] C.-J. Lee, C.-J. Lu, S.-C. Tsai, and W.-G. Tzeng. Extracting randomness from multiple independent sources. *IEEE Transactions on Information Theory*, 51(6):2224–2227, 2005.

[43] A. B. Lewko, M. Lewko, and B. Waters. How to leak on key updates. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM STOC* (ACM Press, 2011), pp. 725–734

[44] T. Malkin, I. Teranishi, Y. Vahlis, and M. Yung. Signatures resilient to continual leakage on memory and computation. In Y. Ishai, editor, *TCC 2011*, vol. 6597 of *LNCS*, (Springer, Berlin, 2011), pp. 89–106

[45] S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In M. Naor, editor, *TCC 2004*, vol. 2951 of *LNCS* (Springer, Berlin, 2004), pp. 278–296

[46] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In S. Halevi, editor, *CRYPTO 2009*, vol. 5677 of *LNCS*, (Springer, Berlin, 2009), pp. 18–35

[47] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In D. B. Shmoys, editor, *46th ACM STOC* (ACM Press, 2014), pp. 475–484

[48] B. Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, (Springer, Berlin, 2009), pp. 619–636

[49] B. Waters. CS 395T Special Topic: Obfuscation in Cryptography. 2014. http://www.cs.utexas.edu/~bwaters/classes/CS395T-Fall-14/outline.html

[50] B. Waters. How to use in distinguishability obfuscation, in *Visions of Cryptography*, 2014. Talk slides available at http://www.cs.utexas.edu/~bwaters/presentations/files/how-to-use-IO.ppt.

[51] D. Wichs. Cryptographic resilience to continual information leakage. Ph.D. Thesis, 2011. http://www.ccs.neu.edu/home/wichs/thesis.pdf

Non-Malleable Codes for Partial Functions with Manipulation Detection

Aggelos Kiayias[1], Feng-Hao Liu[2], and Yiannis Tselekounis[1(✉)]

[1] University of Edinburgh, Edinburgh, UK
akiayias@inf.ed.ac.uk, ytselekounis@ed.ac.uk
[2] Florida Atlantic University, Boca Raton, USA
fenghao.liu@fau.edu

Abstract. Non-malleable codes were introduced by Dziembowski, Pietrzak and Wichs (ICS '10) and its main application is the protection of cryptographic devices against tampering attacks on memory. In this work, we initiate a comprehensive study on non-malleable codes for the class of partial functions, that read/write on an arbitrary subset of codeword bits with specific cardinality. Our constructions are efficient in terms of information rate, while allowing the attacker to access asymptotically almost the entire codeword. In addition, they satisfy a notion which is stronger than non-malleability, that we call non-malleability with manipulation detection, guaranteeing that any modified codeword decodes to either the original message or to $\perp$. Finally, our primitive implies All-Or-Nothing Transforms (AONTs) and as a result our constructions yield efficient AONTs under standard assumptions (only one-way functions), which, to the best of our knowledge, was an open question until now. In addition to this, we present a number of additional applications of our primitive in tamper resilience.

1 Introduction

Non-malleable codes (NMC) were introduced by Dziembowski, Pietrzak and Wichs [27] as a relaxation of error correction and error detection codes, aiming to provide strong privacy but relaxed correctness. Informally, non-malleability guarantees that any modified codeword decodes either to the original message or to a completely unrelated one, with overwhelming probability. The definition of non-malleability is simulation-based, stating that for any tampering function f, there exists a simulator that simulates the tampering effect by only accessing f, i.e., without making any assumptions on the distribution of the encoded message.

The main application of non-malleable codes that motivated the seminal work by Dziembowski et al. [27] is the protection of cryptographic implementations

A. Kiayias—Research partly supported by the H2020 project FENTEC (# 780108).
F.-H. Liu—Research supported by the NSF Award #CNS-1657040.
Y. Tselekounis—Research partly supported by the H2020 project PANORAMIX (# 653497).

H. Shacham and A. Boldyreva (Eds.): CRYPTO 2018, LNCS 10993, pp. 577–607, 2018.
https://doi.org/10.1007/978-3-319-96878-0_20

from *active physical attacks* against memory, known as *tampering attacks*. In this setting, the adversary modifies the memory of the cryptographic device, receives the output of the computation, and tries to extract sensitive information related to the private memory. Security against such types of attacks can be achieved by encoding the private memory of the device using non-malleable codes. Besides that, various applications of non-malleable codes have been proposed in subsequent works, such as CCA secure encryption schemes [20] and non-malleable commitments [4].

Due to their important applications, constructing non-malleable codes has received a lot of attention over recent years. As non-malleability against general functions is impossible [27], various subclasses of tampering functions have been considered, such as split-state functions [1–3,26,27,36,37], bit-wise tampering and permutations [4,5,27], bounded-size function classes [32], bounded depth/fan-in circuits [6], space-bounded tampering [29], and others (cf. Sect. 1.4). One characteristic shared by those function classes is that they allow *full access* to the codeword, while imposing structural or computational restrictions to the way the function computes over the input. In this work we initiate a comprehensive study on non-malleability for functions that receive *partial access* over the codeword, which is an important yet overlooked class, as we elaborate below.

The class of partial functions. The class of *partial functions* contains all functions that read/write on an arbitrary subset of codeword bits with specific cardinality. Concretely, let c be a codeword with length ν. For $\alpha \in [0, 1)$, the function class $\mathcal{F}^{\alpha\nu}$ (or $\mathcal{F}^{\alpha}$ for brevity) consists of all functions that operate over any subset of bits of c with cardinality at most $\alpha\nu$, while leaving the remaining bits intact. The work of Cheraghchi and Guruswami [18] explicitly defines this class and uses a subclass (the one containing functions that always touch the first $\alpha\nu$ bits of the codeword) in a negative way, namely as the tool for deriving capacity lower bounds for *information-theoretic* non-malleable codes against split-state functions. Partial functions were also studied implicitly by Faust et al. [32], while aiming for non-malleability against bounded-size circuits.[1]

Even though capacity lower bounds for partial functions have been derived (cf. [18]), our understanding about *explicit* constructions is still limited. Existential results can be derived by the probabilistic method, as shown in prior works [18,27][2], but they do not yield explicit constructions. On the other hand, the capacity bounds do not apply to the computational setting, which could potentially allow more practical solutions. We believe that this is a direction that needs to be explored, as besides the theoretical interest, partial functions is

[1] Specifically, in [32], the authors consider a model where a common reference string (CRS) is available, with length roughly logarithmic in the size of the tampering function class; as a consequence, the tampering function is allowed to read/write the whole codeword while having only partial information over the CRS.

[2] Informally, prior works [18,27] showed existence of non-malleable codes for classes of certain bounded cardinalities. The results cover the class of partial functions.

a natural model that complies with existing attacks that require partial access to the registers of the cryptographic implementation [8,10–12,44].[3]

Besides the importance of partial functions in the active setting, i.e., when the function is allowed to partially *read/write* the codeword, the passive analogue of the class, i.e., when the function is only given *read access* over the codeword, matches the model considered by All-Or-Nothing Transforms (AONTs), which is a notion originally introduced by Rivest [41], providing security guarantees similar to those of leakage resilience: reading an arbitrary subset (up to some bounded cardinality) of locations of the codeword does not reveal the underlying message. As non-malleable codes provide privacy, non-malleability for partial functions is the active analogue of (and in fact implies) AONTs, that find numerous applications [13,14,40,41,43].

Plausibility. At a first glance one might think that partial functions better comply with the framework of error-correction/detection codes (ECC/EDC), as they do not touch the whole codeword. However, if we allow the adversary to access asymptotically almost the entire codeword, it is conceivable it can use this generous *access rate*, i.e., the fraction of the codeword that can be accessed (see below), to create correlated encodings, thus we believe solving non-malleability in this setting is a natural question. Additionally, non-malleability provides simulation based security, which is not considered by ECC/EDC.

We illustrate the separation between the notions using the following example. Consider the set of partial functions that operate either on the right or on the left half of the codeword (the function chooses if it is going to be left or right), and the trivial encoding scheme that on input message s outputs (s, s). The decoder, on input (s, s'), checks if $s = s'$, in which case it outputs s, otherwise it outputs $\perp$. This scheme is clearly an EDC against the aforementioned function class,[4] as the output of the decoder is in $\{s, \perp\}$, with probability 1; however, it is malleable since the tampering function can create encodings whose validity depends on the message. On the other hand, an ECC would provide a trivial solution in this setting, however it requires restriction of the adversarial access fraction to 1/2 (of the codeword); by accessing more than this fraction, the attacker can possibly create invalid encodings depending on the message, as general ECCs do not provide privacy. Thus, the ECC/EDC setting is inapt when aiming for simulation based security in the presence of attackers that access almost the entire codeword. Later in this section, we provide an extensive discussion on challenges of non-malleability for partial functions.

Besides the plausibility and the lack of a comprehensive study, partial functions can potentially allow stronger primitives, as constant functions are excluded from the class. This is similar to the path followed by Jafargholi and Wichs [34], aiming to achieve *tamper detection* (cf. Sect. 1.4) against a class of

[3] The attacks by [8,11,12] require the modification of a single (random) memory bit, while in [10] a single error per each round of the computation suffices. In [44], the attack requires a single faulty byte.

[4] It is not an ECC as the decoder does not know which side has been modified by the tampering function.

functions that implicitly excludes constant functions and the identity function. In this work we prove that this intuition holds, by showing that partial functions allow a stronger primitive that we define as *non-malleability* with *manipulation detection* (MD-NMC), which in addition to simulation based security, it also guarantees that any tampered codeword will either decode to the original message or to $\perp$. Again, and as in the case of ECC/EDC, we stress out that manipulation/tamper-detection codes do not imply MD-NMC, as they do not provide simulation based security (cf. Sect. 1.4).[5]

Given the above, we believe that partial functions is an interesting and well-motivated model. The goal of this work is to answer the following (informally stated) question:

> *Is it possible to construct efficient (high information rate) non-malleable codes for partial functions, while allowing the attacker to access almost the entire codeword?*

We answer the above question in the affirmative. Before presenting our results (cf. Sect. 1.1) and the high level ideas behind our techniques (cf. Sect. 1.2), we identify the several challenges that are involved in tackling the problem.

Challenges. We first define some useful notions used throughout the paper.

- *Information rate*: the ratio of message to codeword length, as the message length goes to infinity.
- *Access rate*: the fraction of the number of bits that the attacker is allowed to access over the total codeword length, as the message length goes to infinity.

The access rate measures the effectiveness of a non-malleable code in the partial function setting and reflects the level of adversarial access to the codeword. In this work, we aim at constructing non-malleable codes for partial functions with high *information rate* and high *access rate*, i.e., both rates should approach 1 simultaneously. Before discussing the challenges posed by this requirement, we first review some known impossibility results. First, non-malleability for partial functions with concrete access rate 1 is impossible, as the function can fully decode the codeword and then re-encode a related message [27]. Second, information-theoretic non-malleable codes with constant information rate (e.g., 0.5) are not possible against partial functions with constant access rate [18][6], and consequently, solutions in the information-theoretic settings such as ECC and Robust Secret Sharing (RSS) do not solve our problem. Based on these facts, in order to achieve our goal, the only path is to explore the computational setting, aiming for access rate at most $1 - \epsilon$, for some $\epsilon > 0$.

At a first glance one might think that non-malleability for partial functions is easier to achieve, compared to other function classes, as partial functions

[5] Clearly, MD-NMC imply manipulation/error-detection codes.

[6] Informally, in [18] (Theorem 5.3) the authors showed that any information-theoretic non-malleable code with a constant access rate and a constant information rate must have a constant distinguishing probability.

cannot touch the whole codeword. Having that in mind, it would be tempting to conclude that existing designs/techniques with minor modifications are sufficient to achieve our goal. However, we will show that this intuition is misleading, by pointing out why prior approaches fail to provide security against partial functions with high access rate.

The current state of the art in the computational setting considers tools such as (Authenticated) Encryption [1,22,24,28,36,37], *non-interactive zero-knowledge* (NIZK) proofs [22,28,30,37], and ℓ-more extractable collision resistant hashes (ECRH) [36], where others use KEM/DEM techniques [1,24]. Those constructions share a common structure, incorporating a short secret key sk (or a short encoding of it), as well as a long ciphertext, e, and a proof π (or a hash value). Now, consider the partial function f that gets full access to the secret key sk and a constant number of bits of the ciphertext e, partially decrypts e and modifies the codeword depending on those bits. Then, it is not hard to see that non-malleability falls apart as the security of the encryption no longer holds. The attack requires access rate only $O((|sk|)/(|sk| + |e| + |\pi|))$, for [22,28,37] and $O(\text{poly}(k)/|s|)$ for [1,24,36]. A similar attack applies to [30], which is in the continual setting.

One possible route to tackle the above challenges, is to use an encoding scheme over the ciphertext, such that partial access over it does not reveal the underlying message.[7] The guarantees that we need from such a primitive resemble the properties of AONTs, however this primitive does not provide security against active, i.e., tampering, attacks. Another approach would be to use Reconstructable Probabilistic Encodings [6], which provide error-correcting guarantees, yet still it is unknown whether we can achieve information rate 1 for such a primitive. In addition, the techniques and tools for protecting the secret key can be used to achieve optimal information rate as they are independent of the underlying message, yet at the same time, they become the weakest point against partial functions with high access rate. Thus, the question is how to overcome the above challenges, allowing access to almost the entire codeword.

In this paper we solve the challenges presented above based on the following observation: in existing solutions the structure of the codeword is fixed and known to the attacker, and independently of the primitives that we use, the only way to resolve the above issues is by hiding the structure via randomization. This requires a structure recovering mechanism that can either be implemented by an "external" source, or otherwise the structure needs to be reflected in the codeword in some way that the attacker cannot exploit. In the present work we implement this mechanism in both ways, by first proposing a construction in the *common reference string* (CRS) model, and then we show how to remove the CRS using slightly bigger alphabets. Refer to Sect. 1.2 for a technical overview.

[7] In the presence of NIZKs we can have attacks with low access rate that read sk, e, and constant number of bits from the proof.

1.1 Our Results

We initiate the study of *non-malleable codes* with *manipulation-detection* (MD-NMC), and we present the first (to our knowledge) construction for this type of codes. We focus on achieving simultaneously high *information rate* and high *access rate*, in the partial functions setting, which by the results of [18], it can be achieved only in the computational setting.

Our contribution is threefold. First, we construct an information rate 1 non-malleable code in the CRS model, with access rate $1 - 1/\Omega(\log k)$, where k denotes the security parameter. Our construction combines Authenticated Encryption together with an inner code that protects the key of the encryption scheme (cf. Sect. 1.2). The result is informally summarized in the following theorem.

Theorem 1.1 (Informal)**.** *Assuming one-way functions, there exists an explicit computationally secure* MD-NMC *in the* CRS *model, with information rate* 1 *and access rate* $1 - 1/\Omega(\log k)$*, where* k *is the security parameter.*

Our scheme, in order to achieve security with error $2^{-\Omega(k)}$, produces codewords of length $|s| + O(k^2 \log k)$, where $|s|$ denotes the length of the message, and uses a CRS of length $O(k^2 \log k \log(|s| + k))$. We note that our construction does not require the CRS to be fully tamper-proof and we refer the reader to Sect. 1.2 for a discussion on the topic.

In our second result we show how to remove the CRS by slightly increasing the size of the alphabet. Our result is a computationally secure MD-NMC in the standard model, achieving information and access rate $1 - 1/\Omega(\log k)$. Our construction is proven secure by a reduction to the security of the scheme presented in Theorem 1.1. Below, we informally state our result.

Theorem 1.2 (Informal)**.** *Assuming one-way functions, there exists an explicit, computationally secure* MD-NMC *in the standard model, with alphabet length* $O(\log k)$*, information rate* $1 - 1/\Omega(\log k)$ *and access rate* $1 - 1/\Omega(\log k)$*, where* k *is the security parameter.*

Our scheme produces codewords of length $|s|(1 + 1/O(\log k)) + O(k^2 \log^2 k)$.

In Sect. 1.2, we consider security against continuous attacks. We show how to achieve a weaker notion of continuous security, while avoiding the use of a self-destruct mechanism, which was originally achieved by [28]. Our notion is weaker than full continuous security [30], since the codewords need to be updated. Nevertheless, our update operation is deterministic and avoids the full re-encoding process [27,37]; it uses only shuffling and refreshing operations, i.e., we avoid cryptographic computations such as group operations and NIZKs. We call such an update mechanism a "light update." Informally, we prove the following result.

Theorem 1.3 (Informal)**.** *One-way functions imply continuous non-malleable codes with deterministic light updates and without self-destruct, in the standard model, with alphabet length* $O(\log k)$*, information rate* $1 - 1/\Omega(\log k)$ *and access rate* $1 - 1/\Omega(\log k)$*, where* k *is the security parameter.*

As we have already stated, non-malleable codes against partial functions imply AONTs [41]. The first AONT was presented by Boyko [13] in the random oracle model, and then Canetti et al. [14] consider AONTs with public/private parts as well as a secret-only part, which is the full notion. Canetti et al. [14] provide efficient constructions for both settings, yet the fully secure AONT (called "secret-only" in that paper) is based on non-standard assumptions.[8]

Assuming one-way functions, our results yield efficient, fully secure AONTs, in the standard model. This resolves, the open question left in [14], where the problem of constructing AONT under standard assumptions was posed. Our result is presented in the following theorem.

Theorem 1.4 (Informal)**.** *Assuming one-way functions, there exists an explicit secret-only* AONT *in the standard model, with information rate* 1 *and access rate* $1 - 1/\Omega(\log k)$*, where* k *is the security parameter.*

The above theorem is derived by the Informal Theorem 1.1 yielding an AONT whose output consists of both the CRS and the codeword produced by the NMC scheme in the CRS model. A similar theorem can be derived with respect to the Informal Theorem 1.2. Finally, and in connection to AONTs that provide leakage resilience, our results imply leakage-resilient codes [37] for partial functions.

In the full version of the paper we provide concrete instantiations of our constructions, using textbook instantiations [35] for the underlying authenticated encryption scheme. For completeness, we also provide information theoretic variants of our constructions that maintain high access rate and thus necessarily sacrifice information rate.

1.2 Technical Overview

On the manipulation detection property. In the present work we exploit the fact that the class of partial functions does not include constant functions and we achieve a notion that is stronger than non-malleability, which we call *non-malleability* with *manipulation detection*. We formalize this notion as a strengthening of non-malleability and we show that our constructions achieve this stronger notion. Informally, manipulation detection ensures that any tampered codeword will either decode to the original message or to $\perp$.

A MD-NMC **in the CRS model.** For the exposition of our ideas, we start with a naive scheme (which does not work), and then show how we resolve all the challenges. Let $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ be a (symmetric) authenticated encryption scheme and consider the following encoding scheme: to encode a message s, the encoder computes $(sk||e)$, where $e \leftarrow \mathsf{E}_{sk}(s)$ is the ciphertext and $sk \leftarrow \mathsf{KGen}(1^k)$, is the secret key. We observe that the scheme is secure if the tampering function can only read/write on the ciphertext, e, assuming the authenticity property

[8] In [43] the authors present a deterministic AONT construction that provides weaker security.

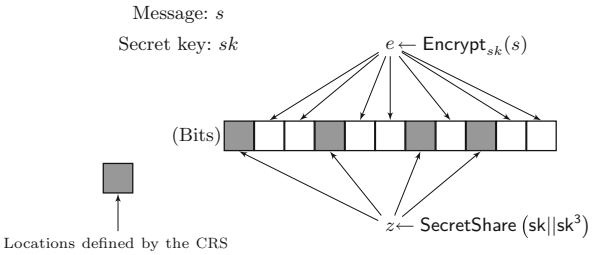

Fig. 1. Description of the scheme in the CRS model.

of the encryption scheme, however, restricting access to sk, which is short, is unnatural and makes the problem trivial. On the other hand, even partial access to sk, compromises the authenticity property of the scheme, and even if there is no explicit attack against the non-malleability property, there is no hope for proving security based on the properties of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$, in black-box way.

A solution to the above problems would be to protect the secret key using an inner encoding, yet the amount of tampering is now restricted by the capabilities of the inner scheme, as the attacker knows the exact locations of the "*sensitive*" codeword bits, i.e., the non-ciphertext bits. In our construction, we manage to protect the secret key while avoiding the bottleneck on the access rate by designing an inner encoding scheme that provides limited security guarantees when used standalone, still when it is used in conjunction with a *shuffling technique* that permutes the inner encoding and ciphertext bit locations, it guarantees that any attack against the secret key will create an invalid encoding with overwhelming probability, even when allowing access to *almost the entire* codeword.

Our scheme is depicted in Fig. 1 and works as follows: on input message s, the encoder (i) encrypts the message by computing $sk \leftarrow \mathsf{KGen}(1^k)$ and $e \leftarrow \mathsf{E}_{sk}(s)$, (ii) computes an m-out-of-m secret sharing z of $(sk||sk^3)$ (interpreting both sk and sk^3 as elements in some finite field),[9] and outputs a random shuffling of $(z||e)$, denoted as $P_\Sigma(z||e)$, according to the common reference string Σ. Decoding proceeds as follows: on input c, the decoder (i) inverts the shuffling operation by computing $(z||e) \leftarrow P_\Sigma^{-1}(c)$, (ii) reconstructs $(sk||sk')$, and (iii) if $sk^3 = sk'$, outputs $\mathsf{D}_{sk}(e)$, otherwise, it outputs $\perp$.

In Sect. 3 we present the intuition behind our construction and a formal security analysis. Our instantiation yields a rate 1 computationally secure MD-NMC in the CRS model, with access rate $1 - 1/\Omega(\log k)$ and codewords of length $|s| + O(k^2 \log k)$, under mild assumptions (e.g., one way functions).

On the CRS. In our work, the tampering function, and consequently the codeword locations that the function is given access to, are fixed before sampling the

[9] In general, any polynomial of small degree, e.g., sk^c, would suffice, depending on the choice of the underlying finite field. Using sk^3 suffices when working over fields of characteristic 2. We could also use sk^2 over fields of characteristic 3.

CRS and this is critical for achieving security. However, proving security in this setting is non-trivial. In addition, the tampering function receives full access to the CRS when tampering with the codeword. This is in contrast to the work by Faust et al. [32] in the information-theoretic setting, where the (internal) tampering function receives partial information over the CRS.

In addition, our results tolerate adaptive selection of the codeword locations, with respect to the CRS, in the following way: each time the attacker requests access to a location, he also learns if it corresponds to a bit of z or e, together with the index of that bit in the original string. In this way, the CRS is gradually disclosed to the adversary while picking codeword locations.

Finally, our CRS sustains a substantial amount of tampering that depends on the codeword locations chosen by the attacker: an attacker that gets access to a sensitive codeword bit is allowed to modify the part of the CRS that defines the location of that bit in the codeword. The attacker is allowed to modify all but $O(k \log(|s| + k))$ bits of the CRS, that is of length $O(k^2 \log k \log(|s| + k))$. To our knowledge, this is the first construction that tolerates, even partial modification of the CRS. In contrast, existing constructions in the CRS model are either using NIZKs [22,28,30,37], or they are based on the *knowledge of exponent assumption* [36], thus tampering access to the CRS might compromise security.

Removing the CRS. A first approach would be to store the CRS inside the codeword together with $P_\Sigma(z||e)$, and give to the attacker read/write access to it. However, the tampering function, besides getting direct (partial) access to the encoding of sk, it also gets indirect access to it by (partially) controlling the CRS. Then, it can modify the CRS in way such that, during decoding, ciphertext locations of its choice will be treated as bits of the inner encoding, z, increasing the tampering rate against z significantly. This makes the task of protecting sk hard, if not impossible (unless we restrict the access rate significantly).

To handle this challenge, we embed the structure recovering mechanism inside the codeword and we emulate the CRS effect by increasing the size of the alphabet, giving rise to a block-wise structure.[10] Notice that, non-malleable codes with large alphabet size (i.e., $\text{poly}(k) + |s|$ bits) might be easy to construct, as we can embed in each codeword block the verification key of a signature scheme together with a secret share of the message, as well as a signature over the share. In this way, partial access over the codeword does not compromise the security of the signature scheme while the message remains private, and the simulation is straightforward. This approach however, comes with a large overhead, decreasing the information rate and access rate of the scheme significantly. In general, and similar to error correcting codes, we prefer smaller alphabet sizes – the larger the size is, the more coarse access structure is required, i.e., in order to access individual bits we need to access the blocks that contain them. In this work, we aim at minimizing this restriction by using small alphabets, as we describe below.

[10] Bigger alphabets have been also considered in the context of error-correcting codes, in which the codeword consists of symbols.

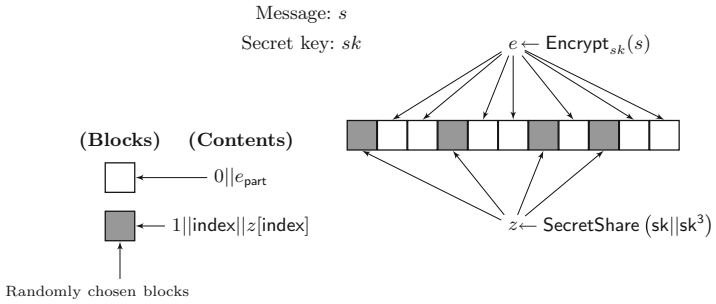

Fig. 2. Description of the scheme in the standard model.

Our approach on the problem is the following. We increase the alphabet size to $O(\log k)$ bits, and we consider two types of blocks: (i) *sensitive blocks*, in which we store the inner encoding, z, of the secret key, sk, and (ii) *non-sensitive* blocks, in which we store the ciphertext, e, that is fragmented into blocks of size $O(\log k)$. The first bit of each block indicates whether it is a sensitive block, i.e., we set it to 1 for sensitive blocks and to 0, otherwise. Our encoder works as follows: on input message s, it computes z, e, as in the previous scheme and then uses rejection sampling to sample the indices, $\rho_1, \ldots, \rho_{|z|}$, for the sensitive blocks. Then, for every $i \in \{1, \ldots, |z|\}$, ρ_i is a sensitive block, with contents $(1||i||z[i])$, while the remaining blocks keep ciphertext pieces of size $O(\log k)$. Decoding proceeds as follows: on input codeword $C = (C_1, \ldots, C_{\mathsf{bn}})$, for each $i \in [\mathsf{bn}]$, if C_i is a non-sensitive block, its data will be part of e, otherwise, the last bit of C_i will be part of z, as it is dictated by the index stored in C_i. If the number of sensitive blocks is not the expected, the decoder outputs $\perp$, otherwise, z, e, have been fully recovered and decoding proceeds as in the previous scheme. Our scheme is depicted in Fig. 2.

The security of our construction is based on the fact that, due to our shuffling technique, the position mapping will not be completely overwritten by the attacker, and as we prove in Sect. 4, this suffices for protecting the inner encoding over sk. We prove security of the current scheme (cf. Theorem 4.4) by a reduction to the security of the scheme in the CRS model. Our instantiation yields a rate $1 - 1/\Omega(\log k)$ MD-NMC in the standard model, with access rate $1-1/\Omega(\log k)$ and codewords of length $|s|(1+1/O(\log k))+O(k^2 \log^2 k)$, assuming one-way functions.

It is worth pointing out that the idea of permuting blocks containing sensitive and non-sensitive data was also considered by [42] in the context of list-decodable codes, however the similarity is only in the fact that a permutation is being used at some point in the encoding process, and our objective, construction and proof are different.

Continuously non-malleable codes with light updates. We observe that the codewords of the block-wise scheme can be updated efficiently, using shuffling and refreshing operations. Based on this observation, we prove that our code is

secure against continuous attacks, for a notion of security that is weaker than the original one [30], as we need to update our codeword. However, our update mechanism is using cheap operations, avoiding the full decoding and re-encoding of the message, which is the standard way to achieve continuous security [27,37]. In addition, our solution avoids the usage of a self-destruction mechanism that produces $\perp$ in all subsequent rounds after the first round in which the attacker creates an invalid codeword, which was originally achieved by [28], and makes an important step towards practicality.

The update mechanism works as follows: in each round, it randomly shuffles the blocks and refreshes the randomness of the inner encoding of sk. The idea here is that, due to the continual shuffling and refreshing of the inner encoding scheme, in each round the attacker learns nothing about the secret key, and every attempt to modify the inner encoding, results to an invalid key, with overwhelming probability. Our update mechanism can be made deterministic if we further encode a seed of a PRG together with the secret key, which is similar to the technique presented in [37].

Our results are presented in Sect. 5 (cf. Theorem 5.3), and the rates for the current scheme match those of the one-time secure, block-wise code.

1.3 Applications

Security against passive attackers - AONTs. Regarding the passive setting, our model and constructions find useful application in all settings where AONTs are useful (cf. [13,14,40,41]), e.g., for increasing the security of encryption without increasing the key-size, for improving the efficiency of block ciphers and constructing remotely keyed encryption [13,41], and also for constructing computationally secure secret sharing [40]. Other uses of AONTs are related to optimal asymmetric encryption padding [13].

Security against memory tampering - (Binary alphabets, Logarithmic length CRS). As with every NMC, the most notable application of the proposed model and constructions is when aiming for protecting cryptographic devices against memory tampering. Using our CRS based construction we can protect a large tamperable memory with a small (logarithmic in the message length) tamperproof memory, that holds the CRS.

The construction is as follows. Consider any device performing cryptographic operations, e.g., a smart card, whose memory is initialized when the card is being issued. Each card is initialized with an independent CRS, which is stored in a tamper-proof memory, while the codeword is stored in a tamperable memory. Due to the independency of the CRS values, it is plausible to assume that the adversary is not given access to the CRS prior to tampering with the card; the full CRS is given to the tampering function while it tampers with the codeword during computation. This idea is along the lines of the *only computation leaks information* model [38], where data can only be leaked during computation, i.e., the attacker learns the CRS when the devices performs computations that depend on it. We note that in this work we allow the tampering function to read

the full CRS, in contrast to [32], in which the tampering function receives partial information over it (our CRS can also be tampered, cf. the above discussion). In subsequent rounds the CRS and the codeword are being updated by the device, which is the standard way to achieve security in multiple rounds while using a one-time NMC [27].

Security against memory tampering - (Logarithmic length alphabets, no CRS). In modern architectures data is stored and transmitted in chunks, thus our block-wise encoding scheme can provide tamper-resilience in all these settings. For instance, consider the case of arithmetic circuits, having memory consisting of consecutive blocks storing integers. Considering adversaries that access the memory of such circuits in a block-wise manner, is a plausible scenario. In terms of modeling, this is similar to tamper-resilience for arithmetic circuits [33], in which the attacker, instead of accessing individual circuit wires carrying bits, it accesses wires carrying integers. The case is similar for RAM computation where the CPU operates over 32 or 64 bit registers (securing RAM programs using NMC was also considered by [22–24,31]). We note that the memory segments in which the codeword blocks are stored do not have to be physically separated, as partial functions output values that depend on the whole input in which they receive access to. This is in contrast to the split-state setting in which the tampering function tampers with each state independently, and thus the states need to be physically separated.

Security against adversarial channels. In Wiretap Channels [9,39,45] the goal is to communicate data privately against eavesdroppers, under the assumption that the channel between the sender and the adversary is "noisier" than the channel between the sender and the receiver. The model that we propose and our block-wise construction can be applied in this setting to provide privacy against a wiretap adversary under the assumption that due to the gap of noise there is a small (of rate $o(1)$) fraction of symbols that are delivered intact to the receiver and dropped from the transmission to the adversary. This enables private, key-less communication between the parties, guaranteeing that the receiver will either receive the original message, or $\perp$. In this way, the communication will be non-malleable in the sense that the receiver cannot be lead to output $\perp$ depending on any property of the plaintext. Our model allows the noise in the receiver side to depend on the transmission to the wiretap adversary, that tampers with a large (of rate $1-o(1)$) fraction of symbols, leading to an "active" variant of the wiretap model.

1.4 Related Work

Manipulation detection has been considered independently of the notion of non-malleability, in the seminal paper by Cramer et al. [21], who introduced the notion of *algebraic manipulation detection* (AMD) codes, providing security against additive attacks over the codeword. A similar notion was considered by Jafargholi and Wichs [34], called *tamper detection*, aiming to detect malicious modifications over the codeword, independently of how those affect the

output of the decoder. Tamper detection ensures that the application of any (admissible) function to the codeword leads to an invalid decoding.

Non-malleable codes for other function classes have been extensively studied, such as constant split-state functions [17,25], block-wise tampering [15,19], while the work of [2] develops beautiful connections among various function classes. In addition, other variants of non-malleable codes have been proposed, such as continuous non-malleable codes [30], augmented non-malleable codes [1], locally decodable/updatable non-malleable codes [16,22–24,31], and non-malleable codes with split-state refresh [28]. In [7] the authors consider AC0 circuits, bounded-depth decision trees and streaming, space-bounded adversaries. Leakage resilience was also considered as an additional feature, e.g., by [16,24,28,37].

2 Preliminaries

In this section we present basic definitions and notation that will be used throughout the paper.

Definition 2.1 (Notation)**.** *Let t, i, j, be non-negative integers. Then, $[t]$ is the set $\{1,\ldots,t\}$. For bit-strings x, y, $x||y$, is the concatenation of x, y, $|x|$ denotes the length of x, for $i \in [|x|]$, $x[i]$ is the i-th bit of x, $||_{j=1}^{t} x_j := x_1||\ldots||x_t$, and for $i \leq j$, $x[i:j] = x[i]||\ldots||x[j]$. For a set I, $|I|$, $\mathcal{P}(I)$, are the cardinality and power set of I, respectively, and for $I \subseteq [|x|]$, $x_{|_I}$ is the projection of the bits of x with respect to I. For a string variable c and value v, $c \leftarrow v$ denotes the assignment of v to c, and $c[I] \leftarrow v$, denotes an assignment such that $c_{|_I}$ equals v. For a distribution D over a set $\mathcal{X}$, $x \leftarrow D$, denotes sampling an element $x \in \mathcal{X}$, according to D, $x \leftarrow \mathcal{X}$ denotes sampling a uniform element x from $\mathcal{X}$, $U_{\mathcal{X}}$ denotes the uniform distribution over $\mathcal{X}$ and $x_1,\ldots,x_t \xleftarrow{\text{rs}} \mathcal{X}$ denotes sampling a uniform subset of $\mathcal{X}$ with t distinct elements, using rejection sampling. The statistical distance between two random variables X, Y, is denoted by $\Delta(X,Y)$, "$\approx$" and "$\approx_c$", denote statistical and computational indistinguishability, respectively, and $\mathsf{negl}(k)$ denotes an unspecified, negligible function, in k.*

Below, we define coding schemes, based on the definitions of [27,37].

Definition 2.2 (Coding scheme [27])**.** *A (κ,ν)-coding scheme, $\kappa,\nu \in \mathbb{N}$, is a pair of algorithms $(\mathsf{Enc},\mathsf{Dec})$ such that: $\mathsf{Enc} : \{0,1\}^\kappa \to \{0,1\}^\nu$ is an encoding algorithm, $\mathsf{Dec} : \{0,1\}^\nu \to \{0,1\}^\kappa \cup \{\perp\}$ is a decoding algorithm, and for every $s \in \{0,1\}^\kappa$, $\Pr[\mathsf{Dec}(\mathsf{Enc}(s)) = s] = 1$, where the probability runs over the randomness used by $(\mathsf{Enc},\mathsf{Dec})$.*

We can easily generalize the above definition for larger alphabets, i.e., by considering $\mathsf{Enc} : \{0,1\}^\kappa \to \Gamma^\nu$ and $\mathsf{Dec} : \Gamma^\nu \to \{0,1\}^\kappa \cup \{\perp\}$, for some alphabet Γ.

Definition 2.3 (Coding scheme in the Common Reference String (CRS) Model [37])**.** *A (κ,ν)-coding scheme in the CRS model, $\kappa,\nu \in \mathbb{N}$,*

is a triple of algorithms $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *such that:* Init *is a randomized algorithm which receives* 1^k*, where* k *denotes the security parameter, and produces a common reference string* $\Sigma \in \{0,1\}^{\mathrm{poly}(k)}$*, and* $(\mathsf{Enc}(1^k, \Sigma, \cdot), \mathsf{Dec}(1^k, \Sigma, \cdot))$ *is a* (κ, ν)*-coding scheme,* $\kappa, \nu = \mathrm{poly}(k)$*.*

For brevity, 1^k will be omitted from the inputs of Enc and Dec.

Below we define *non-malleable codes* with *manipulation detection*, which is a stronger notion than the one presented in [27], in the sense that the tampered codeword will always decode to the original message or to $\perp$. Our definition is with respect to alphabets, as in Sect. 4 we consider alphabets of size $O(\log k)$.

Definition 2.4 (Non-Malleability with Manipulation Detection (MD-NMC))**.** *Let* Γ *be an alphabet, let* $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be a* (κ, ν)*-coding scheme in the common reference string model, and* $\mathcal{F}$ *be a family of functions* $f : \Gamma^\nu \to \Gamma^\nu$*. For any* $f \in \mathcal{F}$ *and* $s \in \{0,1\}^\kappa$*, define the tampering experiment*

$$\mathsf{Tamper}_s^f := \left\{ \begin{array}{c} \Sigma \leftarrow \mathsf{Init}(1^k), c \leftarrow \mathsf{Enc}(\Sigma, s), \tilde{c} \leftarrow f_\Sigma(c), \tilde{s} \leftarrow \mathsf{Dec}(\Sigma, \tilde{c}) \\ \mathit{Output} : \tilde{s}. \end{array} \right\}$$

which is a random variable over the randomness of Enc*,* Dec *and* Init*. The coding scheme* $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *is non-malleable with manipulation detection with respect to the function family* $\mathcal{F}$*, if for all, sufficiently large* k *and for all* $f \in \mathcal{F}$*, there exists a distribution* $D_{(\Sigma,f)}$ *over* $\{0,1\}^\kappa \cup \{\perp, \mathsf{same}^*\}$*, such that for all* $s \in \{0,1\}^\kappa$*, we have:*

$$\left\{\mathsf{Tamper}_s^f\right\}_{k\in\mathbb{N}} \approx \left\{ \begin{array}{c} \tilde{s} \leftarrow D_{(\Sigma,f)} \\ \mathit{Output}\ s\ \mathit{if}\ \tilde{s} = \mathsf{same}^*,\ \mathit{and}\ \perp\ \mathit{otherwise} \end{array} \right\}_{k\in\mathbb{N}}$$

where $\Sigma \leftarrow \mathsf{Init}(1^k)$ *and* $D_{(\Sigma,f)}$ *is efficiently samplable given access to* f*,* Σ*. Here, "*$\approx$*" may refer to statistical, or computational, indistinguishability.*

In the above definition, f is parameterized by Σ to differentiate tamper-proof input, i.e., Σ, from tamperable input, i.e., c.

Below we define the tampering function class that will be used throughout the paper.

Definition 2.5 (The class of partial functions $\mathcal{F}_\Gamma^{\alpha\nu}$ (or $\mathcal{F}^\alpha$))**.** *Let* Γ *be an alphabet,* $\alpha \in [0,1)$ *and* $\nu \in \mathbb{N}$*. Any* $f \in \mathcal{F}_\Gamma^{\alpha\nu}$*,* $f : \Gamma^\nu \to \Gamma^\nu$*, is indexed by a set* $I \subseteq [\nu]$*,* $|I| \leq \alpha\nu$*, and a function* $f' : \Gamma^{\alpha\nu} \to \Gamma^{\alpha\nu}$*, such that for any* $x \in \Gamma^\nu$*,* $(f(x))_{|_I} = f'\left(x_{|_I}\right)$ *and* $(f(x))_{|_{I^c}} = x_{|_{I^c}}$*, where* $I^c := [\nu] \backslash I$*.*

For simplicity, in the rest of the text we will use the notation $f(x)$ and $f(x_{|_I})$ (instead of $f'\left(x_{|_I}\right)$). Also, the length of the codeword, ν, according to Γ, will be omitted from the notation and whenever Γ is omitted we assume that $\Gamma = \{0,1\}$. In Sect. 3, we consider $\Gamma = \{0,1\}$, while in Sect. 4, $\Gamma = \{0,1\}^{O(\log k)}$, i.e., the tampering function operates over blocks of size $O(\log k)$. When considering the CRS model, the functions are parameterized by the common reference string.

The following lemma is useful for proving security throughout the paper.

Lemma 2.6. *Let* $(\mathsf{Enc}, \mathsf{Dec})$ *be a* (κ, ν)*-coding scheme and* $\mathcal{F}$ *be a family of functions. For every* $f \in \mathcal{F}$ *and* $s \in \{0,1\}^\kappa$*, define the tampering experiment*

$$\mathsf{Tamper}_s^f := \left\{ \begin{array}{c} c \leftarrow \mathsf{Enc}(s), \tilde{c} \leftarrow f(c), \tilde{s} \leftarrow \mathsf{Dec}(\tilde{c}) \\ \textit{Output } \mathsf{same}^* \textit{ if } \tilde{s} = s, \textit{ and } \tilde{s} \textit{ otherwise.} \end{array} \right\}$$

which is a random variable over the randomness of Enc *and* Dec*.* $(\mathsf{Enc}, \mathsf{Dec})$ *is an* MD-NMC *with respect to* $\mathcal{F}$*, if for any* $f \in \mathcal{F}$ *and all sufficiently large* k*: (i) for any pair of messages* s_0*,* $s_1 \in \{0,1\}^\kappa$*,* $\left\{\mathsf{Tamper}_{s_0}^f\right\}_{k\in\mathbb{N}} \approx \left\{\mathsf{Tamper}_{s_1}^f\right\}_{k\in\mathbb{N}}$*, and (ii) for any* s*,* $\Pr\left[\mathsf{Tamper}_s^f \notin \{\bot, s\}\right] \leq \mathsf{negl}(k)$*. Here, "*$\approx$*" may refer to statistical, or computational, indistinguishability.*

The proof of the above lemma is provided in the full version of the paper. For coding schemes in the CRS model the above lemma is similar, and Tamper_s^f internally samples $\Sigma \leftarrow \mathsf{Init}(1^k)$.

3 An MD-NMC for Partial Functions, in the CRS Model

In this section we consider $\Gamma = \{0,1\}$ and we construct a rate 1 MD-NMC for $\mathcal{F}^\alpha$, with access rate $\alpha = 1 - 1/\Omega(\log k)$. Our construction is defined below and depicted in Fig. 1.

Construction 3.1. *Let* k*,* $m \in \mathbb{N}$*, let* $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ *be a symmetric encryption scheme,* $(\mathsf{SS}_m, \mathsf{Rec}_m)$ *be an* m*-out-of-*m *secret sharing scheme, and let* $l \leftarrow 2m|sk|$*, where* sk *follows* $\mathsf{KGen}(1^k)$*. We define an encoding scheme* $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$*, that outputs* $\nu = l + |e|$ *bits,* $e \leftarrow \mathsf{E}_{sk}(s)$*, as follows:*

- $\mathsf{Init}(1^k)$*: Sample* $r_1, \ldots, r_l \xleftarrow{\mathsf{rs}} \{0,1\}^{\log(\nu)}$*, and output* $\Sigma = (r_1, \ldots, r_l)$*.*
- $\mathsf{Enc}(\Sigma, \cdot)$*: for input message* s*, sample* $sk \leftarrow \mathsf{KGen}(1^k)$*,* $e \leftarrow \mathsf{E}_{sk}(s)$*.*
  - **(Secret share)** *Sample* $z \leftarrow \mathsf{SS}_m(sk||sk^3)$*, where* $z = \|_{i=1}^{2|sk|} z_i$*,* $z \in \{0,1\}^{2m|sk|}$*, and for* $i \in [|sk|]$*,* z_i *(resp.* $z_{|sk|+i}$*) is an* m*-out-of-*m *secret sharing of* $sk[i]$ *(resp.* $sk^3[i]$*).*
  - **(Shuffle)** *Compute* $c \leftarrow P_\Sigma(z||e)$ *as follows:*
    1. **(Sensitive bits)***: Set* $c \leftarrow 0^\nu$*. For* $i \in [l]$*,* $c[r_i] \leftarrow z[i]$*.*
    2. **(Ciphertext bits)***: Set* $i \leftarrow 1$*. For* $j \in [l + |e|]$*, if* $j \notin \{r_p \mid p \in [l]\}$*,* $c[j] \leftarrow e[i]$*,* i++*.*

  Output c*.*
- $\mathsf{Dec}(\Sigma, \cdot)$*: on input* c*, compute* $(z||e) \leftarrow P_\Sigma^{-1}(c)$*,* $(sk||sk') \leftarrow \mathsf{Rec}_m(z)$*, and if* $sk^3 = sk'$*, output* $\mathsf{D}_{sk}(e)$*, otherwise output* $\bot$*.*

The set of indices of z_i *in the codeword will be denoted by* Z_i*.*

In the above we consider all values as elements over $\mathbf{GF}(2^{\mathrm{poly}(k)})$.

Our construction combines authenticated encryption with an inner encoding that works as follows. It interprets sk as an element in the finite field $\mathbf{GF}(2^{|sk|})$ and computes sk^3 as a field element. Then, for each bit of $(sk||sk^3)$, it computes

an m-out-of-m secret sharing of the bit, for some parameter m (we note that elements in $\mathbf{GF}(2^{|sk|})$ can be interpreted as bit strings). Then, by combining the inner encoding with the shuffling technique, we get a encoding scheme whose security follows from the observations that we briefly present below:

- For any tampering function which does not have access to all m shares of a single bit of $(sk||sk^3)$, the tampering effect on the secret key can be expressed essentially as a linear shift, i.e., as $((sk+\delta)||(sk^3+\eta))$ for some $(\delta,\eta) \in \mathbf{GF}(2^{|sk|}) \times \mathbf{GF}(2^{|sk|})$, independent of sk.
- By permuting the locations of the inner encoding and the ciphertext bits, we have that with overwhelming probability any tampering function who reads/writes on a $(1-o(1))$ fraction of codeword bits, will not learn any single bit of $(sk||sk^3)$.
- With overwhelming probability over the randomness of sk and CRS, for non-zero η and δ, $(sk+\delta)^3 \neq sk^3+\eta$, and this property enables us to design a consistency check mechanism whose output is simulatable, without accessing sk.
- The security of the final encoding scheme follows by composing the security of the inner encoding scheme with the authenticity property of the encryption scheme.

Below we present the formal security proof of the above intuitions.

Theorem 3.2. *Let k, $m \in \mathbb{N}$ and $\alpha \in [0,1)$. Assuming $(\mathsf{SS}_m, \mathsf{Rec}_m)$ is an m-out-of-m secret sharing scheme and $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ is* 1-IND-CPA[11] *secure, authenticated encryption scheme, the code of Construction 3.1 is a* MD-NMC *against $\mathcal{F}^{\alpha}$, for any α, m, such that $(1-\alpha)m = \omega(\log(k))$.*

Proof. Let I be the set of indices chosen by the attacker and $I^{\mathsf{c}} = [\nu] \backslash I$, where $\nu = 2m|sk| + |e|$. The tampered components of the codeword will be denoted using the character "˜" on top of the original symbol, i.e., we have $\tilde{c} \leftarrow f(c)$, the tampered secret key sk (resp. sk^3) that we get after executing $\mathsf{Rec}_m(\tilde{z})$ will be denoted by $\tilde{sk}$ (resp. $\tilde{sk}'$). Also the tampered ciphertext will be $\tilde{e}$. We prove the needed using a series of hybrid experiments that are depicted in Fig. 3. Below, we describe the hybrids.

- $\mathsf{Exp}_0^{\Sigma,f,s}$: We prove security of our code using Lemma 2.6, i.e., by showing that (i) for any s_0, s_1, $\mathsf{Tamper}_{s_0}^f \approx \mathsf{Tamper}_{s_1}^f$, and (ii) for any s, $\Pr\left[\mathsf{Tamper}_s^f \notin \{\perp, s\}\right] \leq \mathsf{negl}(k)$, where Tamper_s^f is defined in Lemma 2.6. For any f, s, $\Sigma \leftarrow \mathsf{Init}(1^k)$, the first experiment, $\mathsf{Exp}_0^{\Sigma,f,s}$, matches the experiment Tamper_s^f in the CRS model, i.e., Σ is sampled inside Tamper_s^f.

[11] This is an abbreviations for indistinguishability under chosen plaintext attack, for a single pre-challenge query to the encryption oracle.

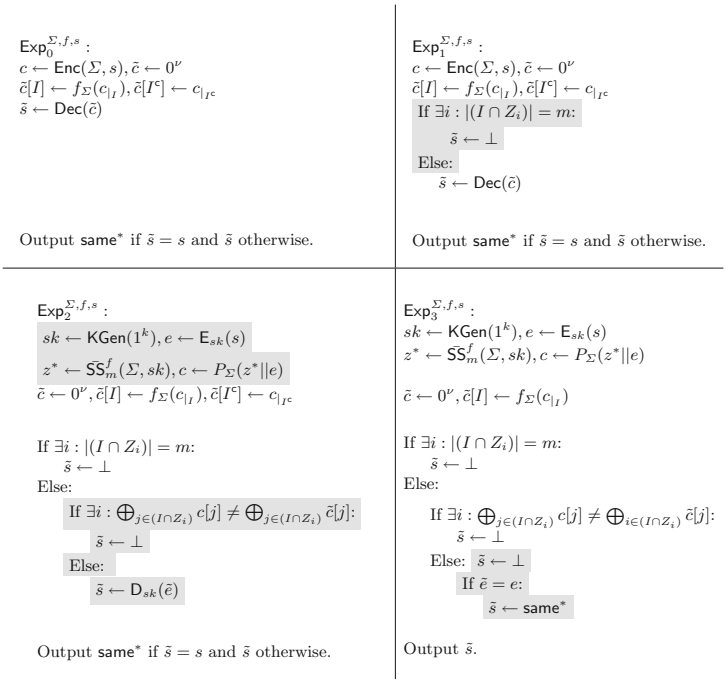

Fig. 3. The hybrid experiments for the proof of Theorem 3.2.

- $\mathsf{Exp}_1^{\Sigma,f,s}$: In the second experiment we define Z_i, $i \in [2|sk|]$, to be the set of codeword indices in which the secret sharing z_i is stored, $|Z_i| = m$. The main difference from the previous experiment is that the current one outputs $\perp$, if there exists a bit of sk or sk^3 for which the tampering function reads all the shares of it, while accessing at most $\alpha\nu$ bits of the codeword. Intuitively, and as we prove in Claim 3.3, by permuting the location indices of $z||e$, this event happens with probability negligible in k, and the attacker does not learn any bit of sk and sk^3, even if he is given access to $(1-o(1))\nu$ bits of the codeword.
- $\mathsf{Exp}_2^{\Sigma,f,s}$: By the previous hybrid we have that for all $i \in [2|sk|]$, the tampering function will not access all bits of z_i, with overwhelming probability. In the third experiment we unfold the encoding procedure, and in addition, we substitute the secret sharing procedure SS_m with $\bar{\mathsf{SS}}_m^f$ that computes shares z_i^* that reveal no information about $sk||sk^3$; for each i, $\bar{\mathsf{SS}}_m^f$ simply "drops" the bit of z_i with the largest index that is not being accessed by f. We formally define $\bar{\mathsf{SS}}_m^f$ below.

$\bar{\mathsf{SS}}_m^f(\Sigma, sk)$:

1. Sample $(z_1, \ldots, z_{2|sk|}) \leftarrow \mathsf{SS}_m\left(sk||sk^3\right)$ and set $z_i^* \leftarrow z_i$, $i \in [2|sk|]$.
2. For $i \in [2|sk|]$, let $l_i := \max_d \{d \in [m] \wedge \mathsf{Ind}(z_i[d]) \notin I)\}$, where Ind returns the index of $z_i[d]$ in c, i.e., l_i is the largest index in $[m]$ such that $z_i[l_i]$ is not accessed by f.
3. (**Output**): For all i set $z_i^*[l_i] = *$, and output $z^* := \|_{i=1}^{2|sk|} z_i^*$.

In $\mathsf{Exp}_1^{\Sigma,f,s}$, $z = \|_{i=1}^{2|sk|} z_i$, and each z_i is an m-out-of-m secret sharing for a bit of sk or sk^3. From Claim 3.3, we have that for all i, $|I \cap Z_i| < m$ with overwhelming probability, and we can observe that the current experiment is identical to the previous one up to the point of computing $f(c_{|_I})$, as $c_{|_I}$ and $f(c_{|_I})$ depend only on z^*, that carries no information about sk and sk^3.

Another difference between the two experiments is in the external "Else" branch: $\mathsf{Exp}_1^{\Sigma,f,s}$ makes a call on the decoder while $\mathsf{Exp}_2^{\Sigma,f,s}$, before calling $\mathsf{D}_{sk}(\tilde{e})$, checks if the tampering function has modified the shares in a way such that the reconstruction procedure $((\tilde{sk}, \tilde{sk}') \leftarrow \mathsf{Rec}_m(\tilde{z}))$ will give $\tilde{sk} \neq sk$ or $\tilde{sk}' \neq sk'$. This check is done by the statement "If $\exists i : \bigoplus_{j \in (I \cap Z_i)} c[j] \neq \bigoplus_{j \in (I \cap Z_i)} \tilde{c}[j]$", without touching sk or sk^3.[12] In case modification is detected the current experiments outputs $\perp$. The intuition is that an attacker that partially modifies the shares of sk and sk^3, creates shares of $\tilde{sk}$ and $\tilde{sk}'$, such that $\tilde{sk}^3 = \tilde{sk}'$, with negligible probability in k. We prove this by a reduction to the 1-IND-CPA security of the encryption scheme: any valid modification over the inner encoding of the secret key gives us method to compute the original secret key sk, with non-negligible probability. The ideas are presented formally in Claim 3.4.

– $\mathsf{Exp}_3^{\Sigma,f,s}$: The difference between the current experiment and the previous one is that instead of calling the decryption $\mathsf{D}_{sk}(\tilde{e})$, we first check if the attacker has modified the ciphertext, in which case the current experiment outputs $\perp$, otherwise it outputs same^*. By the previous hybrid, we reach this newly introduced "Else" branch of $\mathsf{Exp}_3^{\Sigma,f,s}$, only if the tampering function didn't modify the secret key. Thus, the indistinguishability between the two experiments follows from the authenticity property of the encryption scheme in the presence of z^*: given that $\tilde{sk} = sk$ and $\tilde{sk}' = sk'$, we have that if the attacker modifies the ciphertext, then with overwhelming probability $\mathsf{D}_{sk}(\tilde{e}) = \perp$, otherwise, $\mathsf{D}_{sk}(\tilde{e}) = s$, and the current experiment correctly outputs same^* or $\perp$ (cf. Claim 3.5).
– Finally, we prove that for any $f \in \mathcal{F}^\alpha$, and message s, $\mathsf{Exp}_3^{\Sigma,f,s}$ is indistinguishable from $\mathsf{Exp}_3^{\Sigma,f,0}$, where 0 denotes the zero-message. This follows by the semantic security of the encryption scheme, and gives us the indistinguishability property of Lemma 2.6. The manipulation detection property is derived by the indistinguishability between the hybrids and the fact that the output of $\mathsf{Exp}_3^{\Sigma,f,s}$ is in the set $\{\mathsf{same}^*, \perp\}$.

[12] Recall that our operations are over $\mathbf{GF}(2^{\mathrm{poly}(k)})$.

In what follows, we prove indistinguishability between the hybrids using a series of claims.

Claim 3.3. *For $k, m \in \mathbb{N}$, assume $(1-\alpha)m = \omega(\log(k))$. Then, for any $f \in \mathcal{F}^{\alpha}$ and any message s, we have $\mathsf{Exp}_0^{\Sigma,f,s} \approx \mathsf{Exp}_1^{\Sigma,f,s}$, where the probability runs over the randomness used by* Init, Enc.

Proof. The difference between the two experiments is that $\mathsf{Exp}_1^{\Sigma,f,s}$ outputs $\perp$ when the attacker learns all shares of some bit of sk or sk^3, otherwise it produces output as $\mathsf{Exp}_0^{\Sigma,f,s}$ does. Let E the event "$\exists i : |(I \cap Z_i)| = m$". Clearly, $\mathsf{Exp}_0^{\Sigma,f,s} = \mathsf{Exp}_1^{\Sigma,f,s}$ conditioned on $\neg E$, thus the statistical distance between the two experiments is bounded by $\Pr[E]$. In the following we show that $\Pr[E] \leq \mathsf{negl}(k)$. We define by E_i the event in which f learns the entire z_i. Assuming the attacker reads n bits of the codeword, we have that for all $i \in [2|sk|]$,

$$\Pr_{\Sigma}[E_i] = \Pr_{\Sigma}[\ |I \cap Z_i| = m\ ] = \prod_{j=0}^{m-1} \frac{n-j}{\nu-j} \leq \left(\frac{n}{\nu}\right)^m.$$

We have $n = \alpha\nu$ and assuming $\alpha = 1-\epsilon$ for $\epsilon \in (0,1]$, we have $\Pr[E_i] \leq (1-\epsilon)^m \leq 1/e^{m\epsilon}$ and $\Pr[E] = \Pr_{\Sigma}\left[\bigcup_{i=1}^{2|sk|} E_i\right] \leq \frac{2|sk|}{e^{m\epsilon}}$, which is negligible when $(1-\alpha)m = \omega(\log(k))$, and the proof of the claim is complete. ■

Claim 3.4. *Assuming $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ is* 1-IND-CPA *secure, for any $f \in \mathcal{F}^{\alpha}$ and any message s, $\mathsf{Exp}_1^{\Sigma,f,s} \approx \mathsf{Exp}_2^{\Sigma,f,s}$, where the probability runs over the randomness used by* Init, Enc.

Proof. In $\mathsf{Exp}_2^{\Sigma,f,s}$ we unfold the encoding procedure, however instead of calling SS_m, we make a call to $\bar{\mathsf{SS}}_m^f$. As we have already stated above, this modification does not induce any difference between the output of $\mathsf{Exp}_2^{\Sigma,f,s}$ and $\mathsf{Exp}_1^{\Sigma,f,s}$, with overwhelming probability, as z^* is indistinguishable from z in the eyes of f. Another difference between the two experiments is in the external "Else" branch: $\mathsf{Exp}_1^{\Sigma,f,s}$ makes a call on the decoder while $\mathsf{Exp}_2^{\Sigma,f,s}$, before calling $\mathsf{D}_{sk}(\tilde{e})$, checks if the tampering function has modified the shares in a way such that the reconstruction procedure will give $\tilde{sk} \neq sk$ or $\tilde{sk}' \neq sk'$. This check is done by the statement "If $\exists i : \bigoplus_{j\in(I\cap Z_i)} c[j] \neq \bigoplus_{j\in(I\cap Z_i)} \tilde{c}[j]$", without touching sk or sk^3 (cf. Claim 3.3).[13] We define the events E, E' as follows

$$E : \mathsf{Dec}(\tilde{c}) \neq \perp, E' : \exists i : \bigoplus\nolimits_{j\in(I\cap Z_i)} c[j] \neq \bigoplus\nolimits_{j\in(I\cap Z_i)} \tilde{c}[j].$$

Clearly, conditioned on $\neg E'$ the two experiments are identical, since we have $\tilde{sk} = sk$ and $\tilde{sk}' = sk'$, and the decoding process will output $\mathsf{D}_{sk}(\tilde{e})$ in both experiments. Thus, the statistical distance is bounded by $\Pr[E']$. Now, conditioned on $E' \wedge \neg E$, both experiments output $\perp$. Thus, we need to bound

[13] Recall that our operations are over $\mathbf{GF}(2^{\mathrm{poly}(k)})$.

$\Pr[E \wedge E']$. Assuming $\Pr[E \wedge E'] > p$, for $p = 1/\text{poly}(k)$, we define an attacker $\mathcal{A}$ that simulates $\mathsf{Exp}_2^{\Sigma,f,s}$, and uses f, s to break the 1-IND-CPA security of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ in the presence of z^*, with probability at least $1/2 + p''/2$, for $p'' = 1/\text{poly}(k)$.

First we prove that any 1-IND-CPA secure encryption scheme, remains secure even if the attacker receives $z^* \leftarrow \bar{\mathsf{SS}}_m^f(\Sigma, sk)$, as z^* consists of $m-1$ shares of each bit of sk and sk^3, i.e., for the entropy of sk we have $\mathbf{H}(sk|z^*) = \mathbf{H}(sk)$. Towards contradiction, assume there exists $\mathcal{A}$ that breaks the 1-IND-CPA security of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ in the presence of z^*, i.e., there exist s, s_0, s_1 such that $\mathcal{A}$ distinguishes between $(z^*, \mathsf{E}_{sk}(s), \mathsf{E}_{sk}(s_0))$ and $(z^*, \mathsf{E}_{sk}(s), \mathsf{E}_{sk}(s_1))$, with non-negligible probability p. We define an attacker $\mathcal{A}'$ that breaks the 1-IND-CPA security of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ as follows: $\mathcal{A}'$, given $(\mathsf{E}_{sk}(s), \mathsf{E}_{sk}(s_b))$, for some $b \in \{0,1\}$, samples $\hat{sk} \leftarrow \mathsf{KGen}(1^k)$, $\hat{z}^* \leftarrow \bar{\mathsf{SS}}_m^f(\Sigma, \hat{sk})$ and outputs $b' \leftarrow \mathcal{A}(z^*, \mathsf{E}_{sk}(s), \mathsf{E}_{sk}(s_b))$. Since $(z^*, \mathsf{E}_{sk}(s), \mathsf{E}_{sk}(s_b)) \approx (\hat{z}^*, \mathsf{E}_{sk}(s), \mathsf{E}_{sk}(s_b))$ the advantage of $\mathcal{A}'$ in breaking the 1-IND-CPA security of the scheme is the advantage of $\mathcal{A}$ in breaking the 1-IND-CPA security of the scheme in the presence of z^*, which by assumption is non-negligible, and this completes the current proof. We note that the proof idea presented in the current paragraph also applies for proving that other properties that will be used in the rest of the proof, such as semantic security and authenticity, of the encryption scheme, are retained in the presence of z^*.

Now we prove our claim. Assuming $\Pr[E \wedge E'] > p$, for $p = 1/\text{poly}(k)$, we define an attacker $\mathcal{A}$ that breaks the 1-IND-CPA security of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ in the presence of z^*, with non-negligible probability. $\mathcal{A}$ receives the encryption of s, which corresponds to the oracle query right before receiving the challenge ciphertext, the challenge ciphertext $e \leftarrow \mathsf{E}_{sk}(s_b)$, for uniform $b \in \{0,1\}$ and uniform messages s_0, s_1, as well as z^*. $\mathcal{A}$ is defined below.

$$\mathcal{A}\left(z^* \leftarrow \bar{\mathsf{SS}}_m^f(\Sigma, sk), e' \leftarrow \mathsf{E}_{sk}(s), e \leftarrow \mathsf{E}_{sk}(s_b)\right):$$

1. **(Define the shares that will be accessed by f)**: For $i \in [2|sk|]$, define $w_i := (z_i^*)_{|_{[m]\setminus\{l_i\}}}$ and for $i \in [m-1]$ define $C_i = \|_{j=1}^{|sk|} w_j[i]$, $D_i = \|_{j=|sk|+1}^{2|sk|} w_j[i]$.
2. **(Apply f)** Set $c \leftarrow P_\Sigma(z^*||e)$, compute $\tilde{c}[I] \leftarrow f_\Sigma(c_{|_I})$ and let $\tilde{C}_i$, $\tilde{D}_i$, $i \in [m]$, be the tampered shares resulting after the application of f to $c_{|_I}$.
3. **(Guessing the secret key)** Let $U = \sum_{i=1}^{m-1} C_i$, $V = \sum_{i=1}^{m-1} D_i$, i.e., U, V denote the sum of the shares that are being accessed by the attacker (maybe partially), and $\tilde{U} = \sum_{i=1}^{m-1} \tilde{C}_i$, $\tilde{V} = \sum_{i=1}^{m-1} \tilde{D}_i$, are the corresponding tampered values after applying f on U, V. Define

$$p(X) := (U - \tilde{U})X^2 + (U^2 - \tilde{U}^2)X + (U^3 - \tilde{U}^3 - V + \tilde{V}),$$

and compute the set of roots of $p(X)$, denoted as $\mathcal{X}$, which are at most two. Then set

$$\hat{\mathcal{SK}} := \{x + U | x \in \mathcal{X}\}. \tag{1}$$

4. **(Output)** Execute the following steps,
   (a) For $\hat{sk} \in \hat{\mathcal{SK}}$, compute $s' \leftarrow \mathsf{D}_{\hat{sk}}(e')$, and if $s' = s$, compute $s'' \leftarrow \mathsf{D}_{\hat{sk}}(e)$. Output b' such that $s_{b'} = s''$.
   (b) Otherwise, output $b' \leftarrow \{0,1\}$.

In the first step $\mathcal{A}$ removes the dummy symbol "$*$" and computes the shares that will be partially accessed by f, denoted as C_i for sk and as D_i for sk^3. In the second step, it defines the tampered shares, $\tilde{C}_i$, $\tilde{D}_i$. Conditioned on E', it is not hard to see that $\mathcal{A}$ simulates perfectly $\mathsf{Exp}_2^{\Sigma,f,s}$. In particular, it simulates perfectly the input to f as it receives $e \leftarrow \mathsf{E}_{sk}(s)$ and all but $2|sk|$ of the actual bit-shares of sk, sk^3. Part of those shares will be accessed by f. Since for all i, $|I \cap Z_i| < m$, the attacker is not accessing any single bit of sk, sk^3. Let C_m, D_m, be the shares (not provided by the encryption oracle) that completely define sk and sk^3, respectively. By the definition of the encoding scheme and the fact that sk, $sk^3 \in \mathbf{GF}(2^{\mathrm{poly}(k)})$, we have $\sum_{i=1}^{m} C_i = sk$, $\sum_{i=1}^{m} D_i = sk^3$, and

$$(U + C_m)^3 = V + D_m. \tag{2}$$

In order for the decoder to output a non-bottom value, the shares created by the attacker must decode to $\tilde{sk}$, $\tilde{sk}'$, such that $\tilde{sk}^3 = \tilde{sk}'$, or in other words, if

$$\left(\tilde{U} + C_m\right)^3 = \tilde{V} + D_m. \tag{3}$$

From 2 and 3 we receive

$$(U - \tilde{U})C_m^2 + (U^2 - \tilde{U}^2)C_m + (U^3 - \tilde{U}^3) = V - \tilde{V}. \tag{4}$$

Clearly, $\Pr[E \wedge E' \wedge (U = \tilde{U})] = 0$. Thus, assuming $\Pr[E \wedge E'] > p$, for $p > 1/\mathrm{poly}(k)$, we receive

$$\begin{aligned} p < \Pr\left[E \wedge E' \wedge (U \neq \tilde{U})\right] &\leq \Pr\left[\mathsf{Dec}(\tilde{c}) \neq \perp \wedge E' \wedge U \neq \tilde{U}\right] \\ &\leq \Pr\left[\tilde{sk}^3 = \tilde{sk}' \wedge E' \wedge (U \neq \tilde{U})\right] \\ &\overset{(4,1)}{=} \Pr\left[C_m \in \mathcal{X}\right] \overset{(1)}{\leq} \Pr\left[sk \in \hat{\mathcal{SK}}\right], \end{aligned} \tag{5}$$

and $\mathcal{A}$ manages to recover C_m, and thus sk, with non-negligible probability $p' \geq p$. Let W be the event of breaking 1-IND-CPA security. Then,

$$\begin{aligned} \Pr[W] &= \Pr[W|sk \in \hat{\mathcal{SK}}] \cdot \Pr[sk \in \hat{\mathcal{SK}}] \\ &+ \Pr[W|sk \notin \hat{\mathcal{SK}}] \cdot \Pr[sk \notin \hat{\mathcal{SK}}] \\ &\overset{(5)}{=} p' + \frac{1}{2}(1 - p') = \frac{1}{2} + \frac{p'}{2}, \end{aligned} \tag{6}$$

and the attacker breaks the IND-CPA security of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$. Thus, we have $\Pr[E \wedge E'] \leq \mathsf{negl}(k)$, and both experiments output $\perp$ with overwhelming probability. ■

Claim 3.5. *Assuming the authenticity property of* $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$*, for any* $f \in \mathcal{F}^{\alpha}$ *and any message* s, $\mathsf{Exp}_2^{\Sigma,f,s} \approx \mathsf{Exp}_3^{\Sigma,f,s}$*, where the probability runs over the randomness used by* Init, KGen *and* E.

Proof. Before proving the claim, recall that the authenticity property of the encryption scheme is preserved under the presence of z^* (cf. Claim 3.4). Let E be the event $\tilde{sk} = sk \wedge \tilde{sk}' = sk^3$ and E' be the event $\tilde{e} \neq e$. Conditioned on $\neg E$, the two experiments are identical, as they both output $\perp$. Also, conditioned on $E \wedge \neg E'$, both experiments output same^*. Thus, the statistical distance between the two experiments is bounded by $\Pr[E \wedge E']$. Let B be the event $\mathsf{D}_{sk}(\tilde{e}) \neq \perp$. Conditioned on $E \wedge E' \wedge \neg B$ both experiments output $\perp$. Thus, we need to bound $\Pr[E \wedge E' \wedge B]$.

Assuming there exist s, f, for which $\Pr[E \wedge E' \wedge B] > p$, where $p = 1/\mathrm{poly}(k)$, we define an attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that simulates $\mathsf{Exp}_3^{\Sigma,f,s}$ and breaks the authenticity property of the encryption scheme in the presence of z^*, with non-negligible probability. $\mathcal{A}$ is defined as follows: sample $(s, st) \leftarrow \mathcal{A}_1(1^k)$, and then, on input (z^*, e, st), where $e \leftarrow \mathsf{E}_{sk}(s)$, $\mathcal{A}_2$, samples $\Sigma \leftarrow \mathsf{Init}(1^k)$, sets $\tilde{c} \leftarrow 0^{\nu}$, $c \leftarrow P_{\Sigma}(z^*||e)$, computes $\tilde{c}[I] \leftarrow f(c_{|_I})$, $\tilde{c}[I^{\mathsf{c}}] \leftarrow c_{|_{I^{\mathsf{c}}}}$, $(\tilde{z}^*||\tilde{e}) \leftarrow P_{\Sigma}^{-1}(\tilde{c})$, and outputs $\tilde{e}$. Assuming $\Pr[E \wedge E' \wedge B] > p$, we have that $\mathsf{D}_{sk}(\tilde{e}) \neq \perp$ and $\tilde{e} \neq e$, with non-negligible probability and the authenticity property of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ breaks. ∎

Claim 3.6. *Assuming* $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ *is semantically secure, for any* $f \in \mathcal{F}^{\alpha}$ *and any message* s, $\mathsf{Exp}_3^{\Sigma,f,s} \approx \mathsf{Exp}_3^{\Sigma,f,0}$*, where the probability runs over the randomness used by* Init, KGen, E. *"*$\approx$*" may refer to statistical or computational indistinguishability, and* 0 *is the zero-message.*

Proof. Recall that $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ is semantically secure even in the presence of $z^* \leftarrow \bar{\mathsf{SS}}_m^f(\Sigma, sk)$ (cf. Claim 3.4), and towards contradiction, assume there exist $f \in \mathcal{F}^{\alpha}$, message s, and PPT distinguisher D such that

$$\left| \Pr\left[\mathsf{D}\left(\Sigma, \mathsf{Exp}_3^{\Sigma,f,s}\right) = 1\right] - \Pr\left[\mathsf{D}\left(\Sigma, \mathsf{Exp}_3^{\Sigma,f,0}\right)\right] = 1 \right| > p,$$

for $p = 1/\mathrm{poly}(k)$. We are going to define an attacker $\mathcal{A}$ that breaks the semantic security of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ in the presence of z^*, using $s_0 := s$, $s_1 := 0$. $\mathcal{A}$, given z^*, e, executes $\mathsf{Program}$.

$\mathsf{Program}(z^*, e)$:
$c \leftarrow P_{\Sigma}(z^*||e), \tilde{c} \leftarrow 0^{\nu}, \tilde{c}[I] \leftarrow f(c_{|_I})$
If $\exists i : |(I \cap Z_i)| = m$: $\tilde{s} \leftarrow \perp$
Else:
  If $\exists i : \bigoplus_{j \in (I \cap Z_i)} c[j] \neq \bigoplus_{j \in (I \cap Z_i)} \tilde{c}[j]$: $\tilde{s} \leftarrow \perp$
  Else: $\tilde{s} \leftarrow \perp$and If $\tilde{e} = e$: $\tilde{s} \leftarrow \mathsf{same}^*$
Output $\tilde{s}$.

It is not hard to see that $\mathcal{A}$ simulates $\mathsf{Exp}_3^{\Sigma,f,s_b}$, thus the advantage of $\mathcal{A}$ against the semantic security of $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ is the same with the advantage of D in distinguishing between $\mathsf{Exp}_3^{\Sigma,f,s_0}$, $\mathsf{Exp}_3^{\Sigma,f,s_1}$, which by assumption is non-negligible. We have reached a contradiction and the proof of the claim is complete. ∎

From the above claims we have that for any $f \in \mathcal{F}^{\alpha}$ and any s, $\mathsf{Exp}_0^{\Sigma,f,s} \approx \mathsf{Exp}_3^{\Sigma,f,0}$, thus for any $f \in \mathcal{F}^{\alpha}$ and any s_0, s_1, $\mathsf{Exp}_0^{\Sigma,f,s_0} \approx \mathsf{Exp}_0^{\Sigma,f,s_1}$. Also, by the indistinguishability between $\mathsf{Exp}_0^{\Sigma,f,s}$ and $\mathsf{Exp}_3^{\Sigma,f,0}$, the second property of Lemma 2.6 has been proven as the output of $\mathsf{Exp}_3^{\Sigma,f,0}$ is in $\{s, \perp\}$, with overwhelming probability, and non-malleability with manipulation detection of our code follows by Lemma 2.6, since $\mathsf{Exp}_0^{\Sigma,f,s}$ is identical to Tamper_s^f of Lemma 2.6. ■

4 Removing the CRS

In this section we increase the alphabet size to $O(\log(k))$ and we provide a computationally secure, rate 1 encoding scheme in the standard model, tolerating modification of $(1 - o(1))\nu$ blocks, where ν is the total number of blocks in the codeword. Our construction is defined below and the intuition behind it has already been presented in the Introduction (cf. Sect. 1, Fig. 2). In the following, the projection operation will be also used with respect to bigger alphabets, enabling the projection of blocks.

Construction 4.1. *Let k, $m \in \mathbb{N}$, let $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ be a symmetric encryption scheme and $(\mathsf{SS}_m, \mathsf{Rec}_m)$ be an m-out-of-m secret sharing scheme. We define an encoding scheme $(\mathsf{Enc}^*, \mathsf{Dec}^*)$, as follows:*

- $\mathsf{Enc}^*(1^k, \cdot)$*: for input message s, sample $sk \leftarrow \mathsf{KGen}\left(1^k\right)$, $e \leftarrow \mathsf{E}_{sk}(s)$.*
  - **(Secret share)** *Sample $z \leftarrow \mathsf{SS}_m(sk||sk^3)$, where $z = \|_{i=1}^{2|sk|} z_i$, $z \in \{0,1\}^{2m|sk|}$, and for $i \in [|sk|]$, z_i (resp. $z_{|sk|+i}$) is an m-out-of-m secret sharing of $sk[i]$ (resp. $sk^3[i]$).*
  - **(Construct blocks & permute)** *Set $l \leftarrow 2m|sk|$, $\mathsf{bs} \leftarrow \log l + 2$, $d \leftarrow |e|/\mathsf{bs}$, $\mathsf{bn} \leftarrow l + d$, sample $\rho := (\rho_1, \ldots, \rho_l) \xleftarrow{\mathsf{rs}} \{0,1\}^{\log(\mathsf{bn})}$ and compute $C \leftarrow \Pi_\rho(z||e)$ as follows:*
    1. *Set $t \leftarrow 1$, $C_i \leftarrow 0^{\mathsf{bs}}$, $i \in [\mathsf{bn}]$.*
    2. **(Sensitive blocks)** *For $i \in [l]$, set $C_{r_i} \leftarrow (1||i||z[i])$.*
    3. **(Ciphertext blocks)** *For $i \in [\mathsf{bn}]$, if $i \neq r_j$, $j \in [l]$, $C_i \leftarrow (0||e[t : t + (\mathsf{bs} - 1)])$, $t \leftarrow t + (\mathsf{bs} - 1)$.*[14]

  Output $C := (C_1||\ldots||C_{\mathsf{bn}})$.
- $\mathsf{Dec}^*(1^k, \cdot)$*: on input C, parse it as $(C_1||\ldots||C_{\mathsf{bn}})$, set $t \leftarrow 1$, $l \leftarrow 2m|sk|$, $z \leftarrow 0^l$, $e \leftarrow 0$, $\mathcal{L} = \emptyset$ and compute $(z||e) \leftarrow \Pi^{-1}(C)$ as follows:*
  - *For $i \in [\mathsf{bn}]$,*
    - **(Sensitive block)** *If $C_i[1] = 1$, set $j \leftarrow C_i[2 : \mathsf{bs} - 1]$, $z[j] \leftarrow C_i[\mathsf{bs}]$, $\mathcal{L} \leftarrow \mathcal{L} \cup \{j\}$.*
    - **(Ciphertext block)** *Otherwise, set $e[t : t + \mathsf{bs} - 1] = C_i[2 : \mathsf{bs}]$, $t \leftarrow t + \mathsf{bs} - 1$.*
  - *If $|\mathcal{L}| \neq l$, output $\perp$, otherwise output $(z||e)$.*

[14] Here we assume that $\mathsf{bs} - 1$, divides the length of the ciphertext e. We can always achieve this property by padding the message s with zeros, if necessary.

If $\Pi^{-1}(C) = \perp$*, output* $\perp$*, otherwise, compute* $(sk||sk') \leftarrow \mathsf{Rec}_m(z)$*, and if* $sk^3 = sk'$*, output* $\mathsf{D}_{sk}(e)$*, otherwise output* $\perp$*.*

The set of indices of the blocks in which z_i *is stored will be denoted by* Z_i*.*

We prove security for the above construction by a reduction to the security of Construction 3.1. We note that that our reduction is non-black box with respect to the coding scheme in which security is reduced to; a generic reduction, i.e., non-malleable reduction [2], from the standard model to the CRS model is an interesting open problem and thus out of the scope of this work.

In the following, we consider $\Gamma = \{0,1\}^{O(\log(k))}$. The straightforward way to prove that $(\mathsf{Enc}^*, \mathsf{Dec}^*)$ is secure against $\mathcal{F}^{\alpha}_{\Gamma}$ by a reduction to the security of the bit-wise code of Sect. 3, would be as follows: for any $\alpha \in \{0,1\}$, $f \in \mathcal{F}^{\alpha}_{\Gamma}$ and any message s, we have to define α', $g \in \mathcal{F}^{\alpha'}$, such that the output of the tampered execution with respect to $(\mathsf{Enc}^*, \mathsf{Dec}^*)$, f, s, is indistinguishable from the tampered execution with respect to $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$, g, s, and g is an admissible function for $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$. However, this approach might be tricky as it requires the establishment of a relation between α and α' such that the sensitive blocks that f will receive access to, will be simulated using the sensitive bits accessed by g. Our approach is cleaner: for the needs of the current proof we leverage the power of Construction 3.1, by allowing the attacker to choose adaptively the codeword locations, as long as it does not request to read all shares of the secret key. Then, for every block that is accessed by the block-wise attacker f, the bit-wise attacker g requests access to the locations of the bit-wise code that enable him to fully simulate the input to g. We formally present our ideas in the following sections. In Sect. 4.1 we introduce the function class $\mathcal{F}_{\mathsf{ad}}$ that considers adaptive adversaries with respect to the CRS and we prove security of Construction 3.1 in Corollary 4.3 against a subclass of $\mathcal{F}_{\mathsf{ad}}$, and then, we reduce the security of the block-wise code $(\mathsf{Enc}^*, \mathsf{Dec}^*)$ against $\mathcal{F}^{\alpha}_{\Gamma}$ to the security of Construction 3.1 against $\mathcal{F}_{\mathsf{ad}}$ (cf. Sect. 4.2).

4.1 Security Against Adaptive Adversaries

In the current section we prove that Construction 3.1 is secure against the class of functions that request access to the codeword adaptively, i.e., depending on the CRS, as long as they access a bounded number of sensitive bits. Below, we formally define the function class $\mathcal{F}_{\mathsf{ad}}$, in which the tampering function picks up the codeword locations depending on the CRS, and we consider $\Gamma = \{0,1\}$.

Definition 4.2 (The function class $\mathcal{F}^{\nu}_{\mathsf{ad}}$)**.** *Let* $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be an* (κ, ν)*-coding scheme and let* $\mathbb{\Sigma}$ *be the range of* $\mathsf{Init}(1^k)$*. For any* $g = (g_1, g_2) \in \mathcal{F}^{\nu}_{\mathsf{ad}}$*, we have* $g_1 : \mathbb{\Sigma} \rightarrow \mathcal{P}([\nu])$*,* $g_2^{\Sigma} : \{0,1\}^{|\mathsf{range}(g_1)|} \rightarrow \{0,1\}^{|\mathsf{range}(g_1)|} \cup \{\perp\}$*, and for any* $c \in \{0,1\}^{\nu}$*,* $g^{\Sigma}(c) = g_2\left(c_{|_{g_1(\Sigma)}}\right)$*. For brevity, the function class will be denoted as* $\mathcal{F}_{\mathsf{ad}}$*.*

Construction 3.1 remains secure against functions that receive full access to the ciphertext, as well as they request to read all but one shares for each bit of

sk and sk^3. The result is formally presented in the following corollary and its proof, which is along the lines of the proof of Theorem 3.2, is given in the full version of the paper.

Corollary 4.3. *Let k, $m \in \mathbb{N}$. Assuming $(\mathsf{SS}_m, \mathsf{Rec}_m)$ is an m-out-of-m secret sharing scheme and $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ is* 1-IND-CPA *secure authenticated encryption scheme, the code of Construction 4.1 is a* MD-NMC *against any $g = (g_1, g_2) \in \mathcal{F}_{\mathsf{ad}}$, assuming that for all $i \in [2|sk|]$, $(Z_i \cap g_1(\Sigma)) < m$, where $sk \leftarrow \mathsf{KGen}(1^k)$ and $\Sigma \leftarrow \mathsf{Init}(1^k)$.*

4.2 MD-NM Security of the Block-Wise Code

In the current section we prove security of Construction 4.1 against $\mathcal{F}^{\alpha}_{\Gamma}$, for $\Gamma = \{0,1\}^{O(\log(k))}$.

Theorem 4.4. *Let k, $m \in \mathbb{N}$, $\Gamma = \{0,1\}^{O(\log(k))}$ and $\alpha \in [0,1)$. Assuming $(\mathsf{SS}_m, \mathsf{Rec}_m)$ is an m-out-of-m secret sharing scheme and $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ is a* 1-IND-CPA *secure authenticated encryption scheme, the code of Construction 4.1 is a* MD-NMC *against $\mathcal{F}^{\alpha}_{\Gamma}$, for any α, m, such that $(1-\alpha)m = \omega(\log(k))$.*

$g_1(\Sigma = (r_1, \ldots, r_l))$:

- **(Simulate block shuffling)**:
  Sample $\rho := (\rho_1, \ldots, \rho_l) \xleftarrow{\text{rs}} \{0,1\}^{\log(\mathsf{bn})}$
- **(Construct I)**: Set $I = \emptyset$,
  - **(Add ciphertext locations to I)**:
    For $j \in [|e| + l]$, if $j \notin \{r_i | i \in [l]\}$, $I \leftarrow (I \cup j)$.
  - **(Add sensitive bit locations to I according to I_{b})**:
    For $j \in [\mathsf{bn}]$, if $j \in I_{\mathsf{b}}$ and $\exists i \in [l]$ such that $j = \rho_i$, $I \leftarrow (I \cup r_i)$.
- **Output**: Output I.

Fig. 4. The function g_1 that appears in the hybrid experiments of Fig. 7.

$g_2^{\Sigma}(c_{|I})$:
$t \leftarrow 1$, $C_i^* \leftarrow 0^{\mathsf{bs}}$, $i \in [\mathsf{bn}]$.

- **(Reconstruct I)**: Compute $I \leftarrow g_1(\Sigma)$.
- **(Simulate ciphertext blocks)**:
  For $i \in [\mathsf{bn}]$, if $i \neq \rho_j$, $j \in [l]$, $C_i^* \leftarrow (0||e[t : t + (\mathsf{bs} - 1)])$, $t \leftarrow t + (\mathsf{bs} - 1)$.
- **(Simulate sensitive blocks)**:
  - For $i \in [|I|]$, if $\exists j \in [l]$, such that $\mathsf{Ind}(c_{|I}[i]) = r_j$, set $C^*_{\rho_j} \leftarrow \left(1||j||c_{|I}[i]\right)$.
  - Set $C^* := (C_1^*||\ldots||C_{\mathsf{bn}}^*)$ and $\tilde{C}^* := C^*$.
- **(Apply f)**: compute $\tilde{C}^*[I_{\mathsf{b}}] \leftarrow f(C^*_{|I_{\mathsf{b}}})$.
- **(Output)**: Output $\tilde{C}^*_{|I_{\mathsf{b}}}$.

Fig. 5. The function g_2 that appears in the hybrid experiments of Fig. 7.

Proof. Following Lemma 2.6, we prove that for any $f \in \mathcal{F}_{\Gamma}^{\alpha}$, and any pair of messages s_0, s_1, $\mathsf{Tamper}_{s_0}^{f} \approx \mathsf{Tamper}_{s_1}^{f}$, and for any s, $\Pr\left[\mathsf{Tamper}_{s}^{f} \notin \{\perp, s\}\right] \leq \mathsf{negl}(k)$, where Tamper denotes the experiment defined in Lemma 2.6 with respect to the encoding scheme of Construction 4.1, $(\mathsf{Enc}^*, \mathsf{Dec}^*)$. Our proof is given by a series of hybrids depicted in Fig. 7. We reduce the security $(\mathsf{Enc}^*, \mathsf{Dec}^*)$, to the security of Construction 3.1, $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$, against $\mathcal{F}_{\mathsf{ad}}$ (cf. Corollary 4.3). The idea is to move from the tampered execution with respect to $(\mathsf{Enc}^*, \mathsf{Dec}^*)$, f, to a tampered execution with respect to $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$, g, such that the two executions are indistinguishable and $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ is secure against g.

Let I_{b} be the set of indices of the blocks that f chooses to tamper with, where $|I_{\mathsf{b}}| \leq \alpha\nu$, and let $l \leftarrow 2m|sk|$, $\mathsf{bs} \leftarrow \log l + 2$, $\mathsf{bn} \leftarrow l + |e|/\mathsf{bs}$. Below we describe the hybrids of Fig. 7.

- $\mathsf{Exp}_0^{f,s}$: The current experiment is the experiment Tamper_{s}^{f}, of Lemma 2.6, with respect to $(\mathsf{Enc}^*, \mathsf{Dec}^*)$, f, s.
- $\mathsf{Exp}_1^{(g_1,g_2),s}$: The main difference between $\mathsf{Exp}_0^{f,s}$ and $\mathsf{Exp}_1^{(g_1,g_2),s}$, is that in the latter one, we introduce the tampering function (g_1, g_2), that operates over codewords of $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ and we modify the encoding steps so that the experiment creates codewords of the bit-wise code $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$. (g_1, g_2) simulates partially the block-wise codeword C, while given partial access to the bit-wise codeword $c \leftarrow \mathsf{Enc}(s)$. As we prove in the full version, it simulates perfectly the tampering effect of f against $C \leftarrow \mathsf{Enc}^*(s)$.
  g_1 operates as follows (cf. Fig. 4): it simulates perfectly the randomness for the permutation of the block-wise code, denoted as ρ, and constructs a set of indices I, such that g_2 will receive access to, and tamper with, $c_{|_I}$. The set I is constructed with respect to the set of blocks I_{b}, that f chooses to read, as well as Σ, that reveals the original bit positions, i.e., the ones before permuting $(z||e)$. g_2 receives $c_{|_I}$, reconstructs I, simulates partially the blocks of the block-wise codeword, C, and applies f on the simulated codeword. The code of g_2 is given in Fig. 5. In the full version we show that g_2, given $c_{|_I}$, simulates perfectly $C_{|_{I_{\mathsf{b}}}}$, which implies that $g_2^{\Sigma}(c_{|_I}) = f(C_{|_{I_{\mathsf{b}}}})$, and the two executions are identical.
- $\mathsf{Exp}_2^{(g_1,g_3),s}$: In the current experiment, we substitute the function g_2 with g_3, and Dec^* with Dec, respectively. By inspecting the code of g_2 and g_3 (cf. Figs. 5 and 6, respectively), we observe that latter function executes the code of the former, plus the "Check labels and simulate $\tilde{c}[I]$" step. Thus the two experiments are identical up to the point of computing $f(C_{|_{I_{\mathsf{b}}}})$. The main idea here is that we want the current execution to be with respect to $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ against (g_1, g_3). Thus, we substitute Dec^* with Dec, and we expand the function g_2 with some extra instructions/checks that are missing from Dec. We name the resulting function as g_3 and we prove that the two executions are identical.
- Finally, we prove that for any f and any s, $\mathsf{Exp}_2^{(g_1,g_3),s} \approx \mathsf{Exp}_2^{(g_1,g_3),0}$ and $\Pr\left[\mathsf{Exp}_2^{(g_1,g_3),s} \notin \{\perp, s\}\right] \leq \mathsf{negl}(k)$. We do so by proving that (g_1, g_3) is

admissible for $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec},)$, i.e., $(g_1, g_3) \in \mathcal{F}_{\mathsf{ad}}$, and g_3 will not request to access more that $m-1$ shares for each bit of sk, sk^3 (cf. Corollary 4.3). This implies security according to Lemma 2.6.

$g_3^{\Sigma}(c_{|_I})$:

$t \leftarrow 1$, $C_i^* \leftarrow 0^{\mathsf{bs}}$, $i \in [\mathsf{bn}]$.

- **(Reconstruct I)**: Compute $I \leftarrow g_1(\Sigma)$.
- **(Simulate ciphertext blocks)**: For $i \in [\mathsf{bn}]$, if $i \neq \rho_j$, $j \in [l]$, $C_i^* \leftarrow (0||e[t : t + (\mathsf{bs} - 1)])$, $t \leftarrow t + (\mathsf{bs} - 1)$.
- **(Simulate sensitive blocks)**:
  - For $i \in [|I|]$, if $\exists j \in [l]$, such that $\mathsf{Ind}(c_{|_I}[i]) = r_j$, set $C^*_{\rho_j} \leftarrow \left(1||j||c_{|_I}[i]\right)$.
  - Set $C^* := (C_1^*|| \ldots ||C^*_{\mathsf{bn}})$ and $\tilde{C}^* := C^*$.
- **(Apply f)**: compute $\tilde{C}^*[I_{\mathsf{b}}] \leftarrow f(C^*_{|_{I_{\mathsf{b}}}})$.
- **(Check labels and simulate $\tilde{c}[I]$)**: If $\Pi^{-1}(\tilde{C}^*) = \perp$, set $d \leftarrow 1$, otherwise set $(\tilde{z}^*||\tilde{e}) \leftarrow \Pi^{-1}(\tilde{C}^*)$, $\tilde{c}^* \leftarrow P_{\Sigma}(\tilde{z}^*||\tilde{e})$.
- **(Output)**: If $d = 1$ output $\perp$, otherwise output $\tilde{c}^*_{|_I}$.

Fig. 6. The function g_3 that appears in the hybrid experiments of Fig. 7.

$\mathsf{Exp}_0^{f,s}$:
$sk \leftarrow \mathsf{KGen}\left(1^k\right), e \leftarrow \mathsf{E}_{sk}(s)$
$z \leftarrow \mathsf{SS}_m(sk||sk^3)$

$\rho := (\rho_1, \ldots, \rho_l) \xleftarrow{\mathsf{rs}} \{0,1\}^{\log(\mathsf{bn})}$
$C \leftarrow \Pi_{\rho}(z||e), \tilde{C} \leftarrow C$
$\tilde{C}[I_{\mathsf{b}}] \leftarrow f(C_{|_{I_{\mathsf{b}}}})$

$\tilde{s} \leftarrow \mathsf{Dec}^*(\tilde{C})$

Output same^* if $\tilde{s} = s$ and $\tilde{s}$ otherwise.

$\mathsf{Exp}_1^{(g_1,g_2),s}$:
$sk \leftarrow \mathsf{KGen}\left(1^k\right), e \leftarrow \mathsf{E}_{sk}(s)$
$z \leftarrow \mathsf{SS}_m(sk||sk^3)$

$\Sigma \leftarrow \mathsf{Init}^*(1^k), c \leftarrow P_{\Sigma}(z||e)$
$I \leftarrow g_1(\Sigma)$
$C \leftarrow \Pi_{\rho}(z||e), \tilde{C} \leftarrow C$
$\tilde{C}[I_{\mathsf{b}}] \leftarrow g_2^{\Sigma}(c_{|_I})$

$\tilde{s} \leftarrow \mathsf{Dec}^*(\tilde{C})$

Output same^* if $\tilde{s} = s$ and $\tilde{s}$ otherwise.

$\mathsf{Exp}_2^{(g_1,g_3),s}$:
$\Sigma \leftarrow \mathsf{Init}^*(1^k)$
$sk \leftarrow \mathsf{KGen}\left(1^k\right), e \leftarrow \mathsf{E}_{sk}(s)$
$z \leftarrow \mathsf{SS}_m(sk||sk^3)$

$c \leftarrow P_{\Sigma}(z||e), \tilde{c} \leftarrow c$
$I \leftarrow g_1(\Sigma)$
$\tilde{c}[I] \leftarrow g_3^{\Sigma}(c_{|_I})$

$\tilde{s} \leftarrow \mathsf{Dec}(\Sigma, \tilde{c})$

Output same^* if $\tilde{s} = s$ and $\tilde{s}$ otherwise.

Fig. 7. The hybrid experiments for the proof of Theorem 4.4.

The indistinguishability between the hybrids is given in the full version of the paper. ■

5 Continuous MD-NMC with Light Updates

In this section we enhance the block-wise scheme of Sect. 4 with an update mechanism, that uses only shuffling and refreshing operations. The resulting code is secure against continuous attacks, for a notion of security that is weaker than the original one [30], as we need to update our codeword. Below we define the update mechanism, which is denoted as Update^*.

Construction 5.1. *Let $k, m \in \mathbb{N}$, $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$, $(\mathsf{SS}_m, \mathsf{Rec}_m)$ be as in Construction 4.1. We define the update procedure, Update^*, for the encoding scheme of Construction 4.1, as follows:*

– $\mathsf{Update}^*(1^k, \cdot)$*: on input* C*, parse it as* $(C_1 || \ldots || C_{\mathsf{bn}})$*, set* $l \leftarrow 2m|sk|$*,* $\hat{\mathcal{L}} = \emptyset$*, and set* $\hat{C} := (\hat{C}_1 || \ldots || \hat{C}_{\mathsf{bn}})$ *to 0.*
  - **(Secret share** $0^{2|sk|}$**)***: Sample* $z \leftarrow \mathsf{SS}_m\left(0^{2|sk|}\right)$*, where* $z = ||_{i=1}^{2|sk|} z_i$*,* $z \in \{0,1\}^{2m|sk|}$*, and for* $i \in [2|sk|]$*,* z_i *is an* m*-out-of-*m *secret sharing of the* 0 *bit.*
  - **(Shuffle & Refresh)***: Sample* $\rho := (\rho_1, \ldots, \rho_l) \xleftarrow{\mathsf{rs}} \{0,1\}^{\log(\mathsf{bn})}$*. For* $i \in [\mathsf{bn}]$*,*
    * **(Sensitive block)** *If* $C_i[1] = 1$*,*
      · **(Shuffle)***: Set* $j \leftarrow C_i[2 : \mathsf{bs} - 1]$*,* $\hat{C}_{\rho_j} \leftarrow C_i$*.*
      · **(Refresh)***: Set* $\hat{C}_{\rho_j}[\mathsf{bs}] \leftarrow \hat{C}_{\rho_j}[\mathsf{bs}] \oplus z[j]$*.*
    * **(Ciphertext block)**
      If $C_i[1] = 0$*, set* $j \leftarrow \min_n \left\{ n \in [\mathsf{bn}] | n \notin \hat{\mathcal{L}}, n \neq \rho_i, i \in [l] \right\}$*, and* $\hat{C}_j \leftarrow C_i$*,* $\hat{\mathcal{L}} \leftarrow \hat{\mathcal{L}} \cup \{j\}$*.*

  Output $\hat{C}$*.*

The following definition of security is along the lines of the one given in [30], adapted to the notion of non-malleability with manipulation detection. Also, after each invocation the codewords are updated, where in our case the update mechanism is only using shuffling and refreshing operations. In addition, there is no need for self-destruct after detecting an invalid codeword [28].

Definition 5.2 (Continuously MD-NMC with light updates)**.** *Let* $\mathsf{CS} = (\mathsf{Enc}, \mathsf{Dec})$ *be an encoding scheme,* $\mathcal{F}$ *be a functions class and* $k, q \in \mathbb{N}$*. Then,* CS *is a* q*-continuously non-malleable (*q-CNM*) code, if for every, sufficiently large* $k \in \mathbb{N}$*, any pair of messages* s_0*,* $s_1 \in \{0,1\}^{\mathrm{poly}(k)}$*, and any* PPT *algorithm* $\mathcal{A}$*,* $\left\{\mathsf{Tamper}^{\mathcal{A}}_{s_0}(k)\right\}_{k\in\mathbb{N}} \approx \left\{\mathsf{Tamper}^{\mathcal{A}}_{s_1}(k)\right\}_{k\in\mathbb{N}}$*, where,*

$\mathsf{Tamper}^{\mathcal{A}}_s(k)$:
$C \leftarrow \mathsf{Enc}(s), \tilde{s} \leftarrow 0$
For $\tau \in [q]$:
  $f \leftarrow \mathcal{A}(\tilde{s}), \tilde{C} \leftarrow f(C), \tilde{s} \leftarrow \mathsf{Dec}(\tilde{C})$
  If $\tilde{s} = s$: $\tilde{s} \leftarrow \mathsf{same}^*$
  $C \leftarrow \mathsf{Update}^*(1^k, C)$
out $\leftarrow \mathcal{A}(\tilde{s})$
Return : *out*

and for each round the output of the decoder is not in $\{s, \perp\}$ *with negligible probability in* k*, over the randomness of* $\mathsf{Tamper}^{\mathcal{A}}_s$*.*

In the full version of the paper we prove the following statement.

Theorem 5.3. *Let* q*,* k*,* m*,* $\in \mathbb{N}$*,* $\Gamma = \{0,1\}^{O(\log(k))}$ *and* $\alpha \in [0,1)$*. Assuming* $(\mathsf{SS}_m, \mathsf{Rec}_m)$ *is an* m*-out-of-*m *secret sharing scheme and* $(\mathsf{KGen}, \mathsf{E}, \mathsf{D})$ *is a* 1-IND-CPA*, authenticated encryption scheme, the scheme of Construction 5.1 is a continuously* MD-NMC *with light updates, against* $\mathcal{F}^{\alpha}_{\Gamma}$*, for any* α*,* m*, such that* $(1-\alpha)m = \omega(\log(k))$*.*

In the above theorem, q can be polynomial (resp. exponential) in k, assuming the underlying encryption scheme is computationally (resp. unconditionally) secure.

References

1. Aggarwal, D., Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Optimal computational split-state non-malleable codes. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 393–417. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_15
2. Aggarwal, D., Dodis, Y., Kazana, T., Obremski, M.: Non-malleable reductions and applications. In: STOC, pp. 459–468 (2015)
3. Aggarwal, D., Dodis, Y., Lovett, S.: Non-malleable codes from additive combinatorics. In: STOC, pp. 774–783 (2014)
4. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Explicit non-malleable codes against bit-wise tampering and permutations. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 538–557. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_26
5. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 375–397. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_16
6. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes for bounded depth, bounded fan-in circuits. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 881–908. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_31
7. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. Cryptology ePrint Archive, Report 2017/1061 (2017)
8. Bao, F., Deng, R.H., Han, Y., Jeng, A., Narasimhalu, A.D., Ngair, T.: Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In: Christianson, B., Crispo, B., Lomas, M., Roe, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, pp. 115–124. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0028164
9. Bellare, M., Tessaro, S., Vardy, A.: Semantic security for the wiretap channel. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 294–311. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_18
10. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052259
11. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_4
12. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. J. Cryptol. **14**(2), 101–119 (2001)
13. Boyko, V.: On the security properties of OAEP as an all-or-nothing transform. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 503–518. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_32

14. Canetti, R., Dodis, Y., Halevi, S., Kushilevitz, E., Sahai, A.: Exposure-resilient functions and all-or-nothing transforms. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 453–469. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_33
15. Chandran, N., Goyal, V., Mukherjee, P., Pandey, O., Upadhyay, J.: Block-wise non-malleable codes. IACR Cryptology ePrint Archive, p. 129 (2015)
16. Chandran, N., Kanukurthi, B., Raghuraman, S.: Information-theoretic local non-malleable codes and their applications. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A. LNCS, vol. 9563, pp. 367–392. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_14
17. Chattopadhyay, E., Zuckerman, D.: Non-malleable codes against constant split-state tampering. In: FOCS, pp. 306–315 (2014)
18. Cheraghchi, M., Guruswami, V.: Capacity of non-malleable codes. In: ITCS 2014 (2014)
19. Choi, S.G., Kiayias, A., Malkin, T.: BiTR: built-in tamper resilience. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 740–758. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_40
20. Coretti, S., Maurer, U., Tackmann, B., Venturi, D.: From single-bit to multi-bit public-key encryption via non-malleable codes. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 532–560. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_22
21. Cramer, R., Dodis, Y., Fehr, S., Padró, C., Wichs, D.: Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 471–488. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_27
22. Dachman-Soled, D., Kulkarni, M., Shahverdi, A.: Locally decodable and updatable non-malleable codes in the bounded retrieval model. Cryptology ePrint Archive, Report 2017/303 (2017). http://eprint.iacr.org/2017/303
23. Dachman-Soled, D., Kulkarni, M., Shahverdi, A.: Tight upper and lower bounds for leakage-resilient, locally decodable and updatable non-malleable codes. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10174, pp. 310–332. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54365-8_13
24. Dachman-Soled, D., Liu, F.-H., Shi, E., Zhou, H.-S.: Locally decodable and updatable non-malleable codes and their applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 427–450. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_18
25. Döttling, N., Nielsen, J.B., Obremski, M.: Information theoretic continuously non-malleable codes in the constant split-state model. Cryptology ePrint Archive, Report 2017/357 (2017). http://eprint.iacr.org/2017/357
26. Dziembowski, S., Kazana, T., Obremski, M.: Non-malleable codes from two-source extractors. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 239–257. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_14
27. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: ICS (2010)
28. Faonio, A., Nielsen, J.B.: Non-malleable codes with split-state refresh. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10174, pp. 279–309. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54365-8_12
29. Faust, S., Hostáková, K., Mukherjee, P., Venturi, D.: Non-malleable codes for space-bounded tampering. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 95–126. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_4

30. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: Continuous non-malleable codes. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 465–488. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_20
31. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: A tamper and leakage resilient von neumann architecture. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 579–603. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_26
32. Faust, S., Mukherjee, P., Venturi, D., Wichs, D.: Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 111–128. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_7
33. Genkin, D., Ishai, Y., Prabhakaran, M.M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: STOC 2014, pp. 495–504 (2014)
34. Jafargholi, Z., Wichs, D.: Tamper detection and continuous non-malleable codes. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 451–480. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_19
35. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (2007)
36. Kiayias, A., Liu, F.-H., Tselekounis, Y.: Practical non-malleable codes from l-more extractable hash functions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 1317–1328. ACM, New York (2016)
37. Liu, F.-H., Lysyanskaya, A.: Tamper and leakage resilience in the split-state model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 517–532. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_30
38. Micali, S., Reyzin, L.: Physically observable cryptography. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_16
39. Ozarow, L.H., Wyner, A.D.: Wire-tap channel II. AT&T Bell Lab. Tech. J. **63**(10), 2135–2157 (1984)
40. Resch, J.K., Plank, J.S.: AONT-RS: blending security and performance in dispersed storage systems. In: FAST 2011 (2011)
41. Rivest, R.L.: All-or-nothing encryption and the package transform. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 210–218. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052348
42. Shaltiel, R., Silbak, J.: Explicit list-decodable codes with optimal rate for computationally bounded channels. In: APPROX/RANDOM 2016 (2016)
43. Stinson, D.R.: Something about all or nothing (transforms). Des. Codes Crypt. **22**(2), 133–138 (2001)
44. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 224–233. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21040-2_15
45. Wyner, A.D.: The wire-tap channel. Bell Syst. Tech. J. **54**(8), 1355–1387 (1975)

Parameter-Hiding Order Revealing Encryption

David Cash[1], Feng-Hao Liu[2], Adam O'Neill[3], Mark Zhandry[4], and Cong Zhang[5(✉)]

[1] Department of Computer Science, University of Chicago, Chicago, USA
davidcash@cs.uchicago.edu
[2] Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, USA
[3] Department of Computer Science, Georgetown University, Washington, D.C., USA
[4] Department of Computer Science, Princeton University, Princeton, USA
[5] Department of Computer Science, Rutgers University, New Brunswick, USA
cz200@cs.rutgers.edu

Abstract. Order-revealing encryption (ORE) is a primitive for outsourcing encrypted databases which allows for efficiently performing range queries over encrypted data. Unfortunately, a series of works, starting with Naveed *et al.* (CCS 2015), have shown that when the adversary has a good estimate of the distribution of the data, ORE provides little protection. In this work, we consider the case that the database entries are drawn identically and independently from a distribution of known shape, but for which the mean and variance are not (and thus the attacks of Naveed *et al.* do not apply). We define a new notion of security for ORE, called *parameter-hiding ORE*, which maintains the secrecy of these parameters. We give a construction of ORE satisfying our new definition from bilinear maps.

Keywords: Encryption · Order-revealing encryption

1 Introduction

An emerging area of cryptography concerns the design and analysis of "leaky" protocols (see *e.g.* [11,33,36] and additional references below), which are protocols that deliberately give up some level of security in order to achieve better efficiency. One important tool in this area is *order-revealing encryption* [7,8][1]. Order-revealing encryption (ORE) is a special type of symmetric encryption which leaks the order of the underlying plaintexts through a *public* procedure Comp. In practice, ORE allows for a client to store a database on an untrusted server in encrypted form, while still permitting the server to efficiently perform various operations such as range queries on the encrypted data without the secret decryption key. ORE has been implemented and used in real-world encrypted database systems, including CryptDB [36].

[1] In [7], it was called efficiently-orderable encryption.

T. Peyrin and S. Galbraith (Eds.): ASIACRYPT 2018, LNCS 11272, pp. 181–210, 2018.
https://doi.org/10.1007/978-3-030-03326-2_7

Various notions of ORE have been proposed. The strongest, called "ideal" ORE, insists that everything about the plaintexts is hidden, except for their order. For example, it should be impossible to distinguish between encryptions of $1, 2, 3$ and $1, 4, 9$. Such ideal ORE can be constructed from multilinear maps [8], showing that in principle ideal ORE is achievable. However, current multilinear maps are quite inefficient, and moreover have been subject to numerous attacks (e.g. [16,17,32]).

In order to develop efficient schemes, one can relax the security requirements to allow for more leakage. Order-preserving encryption (OPE) [1,6]—which actually predates ORE—is one example, where Comp is simply integer comparison. Very efficient constructions of OPE are known [6]. However, OPE necessarily leaks much more information about the plaintexts [6] than ideal ORE; intuitively, the difference between ciphertexts can be used to approximate the difference between the plaintexts. More recently, there have been efforts to achieve better security without sacrificing too much efficiency: Chenette, Lewi, Weis and Wu (CLWW) [15] recently gave an ORE construction which leaks only the position of the most significant differing bits of the plaintexts.

Unfortunately, even hypothetical ideal ORE has recently been shown insecure for various use cases [3,10,19,20,22,24–26,30,34]. This is even if the scheme itself reveals nothing but the order of the plaintexts. The problem is that just the order of plaintexts alone can already reveal a significant amount of information about the data. For example, if the data is chosen uniformly from the entire domain, then even *ideal* ORE will leak the most significant bits. As the most significant bits are often the most important ones, this is troubling.

The problem is that the definitions of ORE, while precise and provable, do not immediately provide any "semantically meaningful" guarantees for the privacy of the underlying data. Indeed, the above attacks show that when the adversary has a strong estimate of the prior distribution the data is drawn from, essentially no security is possible. However, we contend that there are scenarios (see below) where the adversary lacks this knowledge. A core problem in such scenarios is that the privacy of one message is inherently dependent on what other ciphertexts the adversary sees. Analyzing these correlations under arbitrary sources of data, even for ideal ORE, can be quite difficult. Only very mild results are known, for example the fact that either CLWW leakage or ideal leakage provably hides the *least* significant bits of uniformly chosen data. Unfortunately, these bits are probably of less importance (e.g. for salaries).

Therefore, a central goal of this paper is to devise a semantically meaningful notion of privacy for the underlying data in the case that the adversary does not have a strong estimate of the prior distribution, and develop a construction attaining this notion not based on multilinear maps.

We stress that we are not trying to devise a scheme that is secure in the use cases of the attacks above, as many of the attacks above would apply to *any* ORE scheme; we are instead aiming to identify settings where the attacks do not apply, and then provide a scheme satisfying a given notion of security in this setting.

1.1 This Work: Parameter-Hiding ORE

In this work, we give one possible answer to the question above. Rather than focusing on the individual data records, we instead ask about the privacy of the distribution they came from. We show how to protect some information about the underlying data distribution.

Motivating Example. To motivate our notion, consider the following setting. A large university wants to outsource its database of student GPAs. For simplicity, we will assume each student's academic ability is independent of other students, and that this is reflected in the GPA. Thus, we will assume that each GPA is sampled independently and identically according to some underlying distribution. The university clearly wants to keep each individual's GPA hidden. It also may want aggregate statistics such as mean and variance to be hidden, perhaps to avoid getting a reputation for handing out very high or very low grades.

Distribution-Hiding ORE. This example motivates a notion of distribution-hiding ORE, where all data is sampled independently and identically from some underlying distribution D, and we wish to hide as much as possible about D. We would ideally like to handle arbitrary distributions D, but in many cases will accept handling certain special classes of distributions. Notice that if the distribution itself is completely hidden, then so too is every individual record, since any information about a record is also information about D.

We begin with the following trivial observation: if D has high min-entropy (namely, super-logarithmic), then the ideal ORE leakage is just a random ordering with no equalities, since there are no collisions with overwhelming probability. In particular, this leakage is independent of the distribution D; as such, ideal ORE leakage hides everything about the underlying distribution, except for the super-logarithmic lower bound on min-entropy. Thus, we can use the multilinear map-based scheme of [8] to achieve distribution-hiding ORE for any distribution with high min-entropy.

We note the min-entropy requirement is critical, since for smaller min-entropies, the leakage allows for determining the frequency of the most common elements, hence learning non-trivial information about D.[2]

Unfortunately, the only way we know to build distribution-hiding ORE is using ideal leakage as above; as such, we do not know of a construction not based on multilinear maps. Instead, in hopes of building such a scheme, we will allow some information about the distribution to leak.

[2] This min-entropy requirement may be somewhat problematic in some settings. GPAs for example, probably have fewer than 10 bits of entropy. However, adding small random noise to the data before encrypting (much smaller than the precision of the data) will force the data to have high min-entropy without changing the order of data, with the exception that identical data will appear different when comparing. In many cases (such as answering range queries) it is totally acceptable to fail to identify identical data.

Parameter-Hiding ORE. We recall that in many settings, data follows a known type of distribution. For example, the central limit theorem implies that many quantities such as various physical, biological, and financial quantities are (approximately) normally distributed. It is also common practice to assign grades on an approximately normal distribution, so GPAs might reasonably be conjectured to be normal. For a different example, insurance claims are often modeled according to the Gamma distribution.

Therefore, since the general shape of the distribution is typically known, a reasonable relaxation of distribution-hiding ORE is what we will call *parameter-hiding ORE*. Here, we will assume the distribution has a *known, public* "shape" (*e.g.* normal, uniform, Laplace *etc.*) but it may be shifted or scaled. We will allow the overall shape to be revealed; our goal instead is to completely hide the shifting and scaling information. More precisely, we consider a distribution D over $[0, 1]$ which will describe the general shape of the family of distributions in question. For example, if the shape in consideration is the set of uniform distributions over an interval, we may take D to be uniform distribution over $[0, 1]$; if the shape is the normal distribution, we will take D be the normal distribution with mean $1/2$, and standard deviation small enough so that the vast majority of the mass is in $[0, 1]$. Let $D_{\alpha,\beta}$ be the distribution defined as: first sample $x \leftarrow D$, and then output $\lfloor \alpha x + \beta \rfloor$. We will call α the scaling term and β the shift. The adversary receives a polynomial number of encryptions of plaintext sampled iid from $D_{\alpha,\beta}$ for some α, β. We will call a scheme *parameter hiding* if the scale and shift are hidden from any computationally bounded adversary. Our main theorem is that it is possible to construct such parameter-hiding ORE from bilinear maps:

Theorem 1 (Informal). *Assuming bilinear maps, it is possible to construct parameter-hiding ORE for any "smooth" distribution D, provided the scaling term is "large enough."*

We note the restrictions to large scalings are inherent: any small scaling will lead to a distribution with low min-entropy. As discussed above, even with ideal ORE, it is possible to estimate the min-entropy of low min-entropy distributions, and hence it would be possible to recover the scaling term if the scaling term is small. Some restrictions on the shape of D are also necessary, as certain shapes can yield low min-entropy even for large scalings. "Smoothness" (which we will define as having a bounded derivative) guarantees high min-entropy at large scales, and is also important technically for our analysis.

1.2 Technical Overview

As a starting point, we will consider the leakage profile of Chenette, Lewi, Weis and Wu [15] (henceforth referred to as CLWW), which reveals the position of the most significant differing bit between any two plaintexts. This is quite a lot of information: for example, it can be used to get rough bounds on the difference between two plaintexts. Thus, CLWW cannot be parameter hiding, since the scaling term is not hidden. However, CLWW will be a useful starting point, as

it will allow us to construct *shift-hiding* ORE, where we only care about hiding the shift term. To help illustrate our approach, we will therefore first describe an equivalent formulation of CLWW leakage, which we will then explain how to extend to get full parameter-hiding ORE.

An Alternative View of CLWW Leakage. Consider the plaintext space $\{0, 1, 2, \ldots, 2^\ell - 1\}$. We will think of the plaintexts as leaves in a full binary tree of depth ℓ. In this tree, the position of the most significant differing bit between two plaintexts corresponds to the depth of their nearest ancestor. The leakage of CLWW can therefore can be seen as revealing the tree consisting of all given plaintexts, their ancestors in the tree up to the lowest common ancestor, and the order of the leaves, with all other information removed. See Fig. 1 for an illustration.

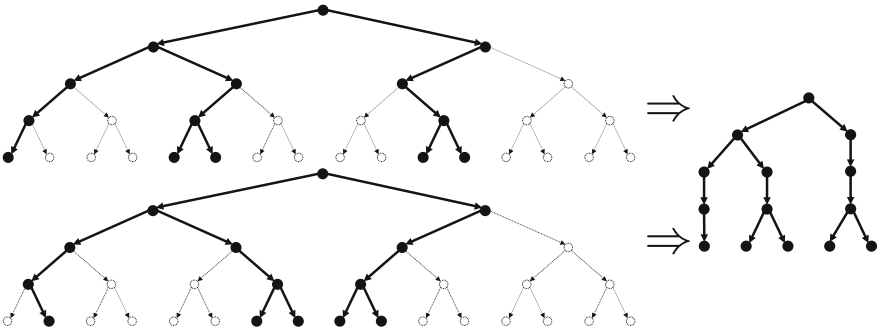

Fig. 1. CLWW Leakage. The two sets of plaintext $\{0, 4, 5, 10, 11\}$ and $\{1, 6, 7, 8, 9\}$ correspond to equivalent subtrees. If the message space extends beyond 15, the CLWW leakage remains the same as depicted, since the leakage only reveals the tree up to the most recent ancestor.

Now, suppose all plaintext elements are in the range $[0, 2^i)$ for some i. This means they all belong in the same subtree at height i; in particular, the CLWW leakage will only have depth at most i. Now, suppose we add a multiple of 2^i to every plaintext. This will simply shift all the plaintexts to being in a different subtree, but otherwise keep the same structure. Therefore, the CLWW leakage will remain the same.

Therefore, while CLWW is not shift hiding, it is *shift periodic*. In particular, if imagine a distribution D whose support is on $[0, 2^i)$, and consider shifting D by β. Consider an adversary A, which is given the CLWW leakage from q plaintexts sampled from the shifted D, and outputs a bit. If we plot the probability $p(\beta)$ that A outputs 1 as a function of β, we will see that the function is periodic with period 2^i.

Shift-Hiding ORE/OPE. With this periodicity, it is simple to construct a scheme that is shift hiding. To get a shift-hiding scheme for message space $[0, 2^\ell)$, we instantiate CLWW with message space $[0, 2^{\ell+1})$. We also include as part of the secret key a random shift γ chosen uniformly in $[0, 2^\ell)$. We then encrypt a message m as $\mathsf{Enc}(m + \gamma)$. Adding a random shift can be seen as convolving the signal $p(\beta)$ with the rectangular function

$$q(\beta) = \begin{cases} 2^{-\ell} & \text{if } \beta \in [0, 2^\ell) \\ 0 & \text{otherwise} \end{cases}$$

Since the rectangular function's support matches the period of p, the result is that the convolved signal $\hat{p}$ is *constant*. In other words, the adversary always has the same output distribution, regardless of the shift β. Thus, we achieve shift hiding.

When the comparison algorithm of an ORE scheme is simple integer comparison, we say the scheme is an *order-preserving encryption* (OPE) scheme. OPE is preferable because it can be used with fewer modifications to a database server. We recall that CLWW can be made into an OPE scheme — where ciphertexts are integers and comparision is integer comparison — while maintaining the CLWW leakage profile. Our conversion to shift-hiding preserves the OPE property, so we similarly achieve a shift-hiding OPE scheme.

Scale-Hiding ORE/OPE. We note that we can also turn any shift-hiding ORE into a scale-hiding ORE. Simply take the logarithm of the input before encrypting; now multiplying by a constant corresponds to shifting by a constant. Of course, taking the logarithm will result in non-integers; this can easily be fixed by rounding to the appropriate level of precision (enough precision to guarantee no collisions over the domain) and scaling up to make the plaintexts integral. Similarly, we can also obtain scale-hiding OPE if we start with an OPE scheme.

Impossibility of Parameter-Hiding OPE. One may hope to achieve both shift-hiding and scale-hiding by some combination of the two above schemes. For example, since order *preserving* encryption schemes can be composed, one can imagine composing a shift-hiding scheme with a scale-hiding scheme. Interestingly, this does not give a parameter-hiding scheme. The reason is that shifts/scalings of the plaintext do not correspond to shifts/scalings of the ciphertexts. Therefore, while the outer OPE may provide, say, shift-hiding for its inputs, this will not translate to shift-hiding of the inner OPE's inputs.

Nonetheless, one may hope that tweaks to the above may give a scheme that is simultaneously scale and shift hiding. Perhaps surprisingly, we show that this is actually impossible. Namely, we show that *OPE cannot possibly be parameter-hiding*. Due to space limit, we put the rigorous proof in our full version [12].

This impossibility shows that strategies leveraging CLWW leakage are unlikely to yield parameter-hiding ORE schemes. Interestingly, all ORE schemes we are aware of that can be constructed from symmetric crypto can also be made into OPE schemes. Thus, this suggests we need stronger tools than those used by previous efficient schemes.

Parameter Hiding via Smoothed CLWW Leakage. Motivated by the above, we must seek a different leakage profile if we are to have any hope of achieving parameter-hiding ORE. We therefore first describe a "dream" leakage that will allow us to perform similar tricks as in the shift hiding case in order to achieve both scale and shift hiding simultaneously. Our dream leakage will be a "smoothed" CLWW leakage, where all nodes of degree exactly 2 are replaced with an edge between the two neighbors. In other words, the dream leakage is the smallest graph that is "homeomorphic" to the CLWW leakage. See Fig. 2 for an illustration.

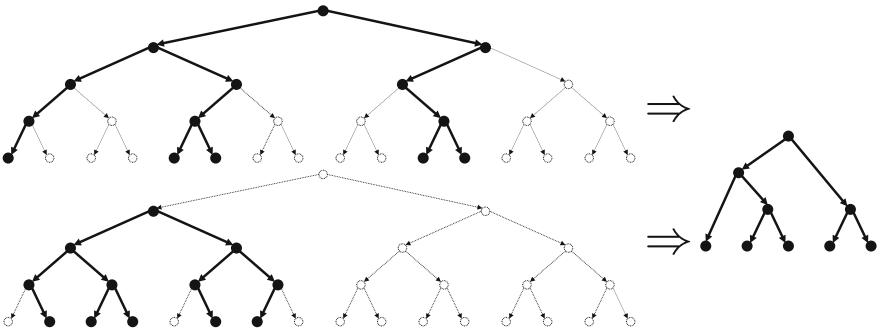

Fig. 2. Smoothed CLWW Leakage. The two sets of plaintext $\{0, 4, 5, 10, 11\}$ and $\{1, 2, 3, 5, 6\}$ correspond to equivalent smoothed subtrees. Notice that the CLWW leakage for these two trees is different.

Our key observation is that this smoothed CLWW leakage now exhibits additional periodicity. Namely, if we multiply every plaintext by 2, every edge in the bottom layer of the CLWW leakage will get subdivided into a path of length 2, but smoothing out the leakage will result in the same exact graph. This means that smoothed CLWW leakage is periodic in the *log* domain.

In particular, consider a distribution D with support on $[0, 2^i)$, and suppose it is multiplied by α. Consider an adversary A, which is given the smoothed CLWW leakage from q plaintexts sampled from a scaled D, and outputs a bit. If we plot the probability $p(\log_2 \alpha)$ that A outputs 1 as a function of α, we will see that the function is periodic with period 1.

Therefore, we can perform a similar trick as above. Namely, we convolve p with the uniform distribution over the period of p in the log domain. We accomplish this by including a random scalar α as part of the secret key, and multiplying by α before encrypting. However, this time several things are different:

- Since we are working in the log domain, the logarithm of the random scalar α has to be uniform. In other words, α is log-uniform
- Since we are working over integers instead of real numbers, many issues arise.

- First, α needs to be an integer to guarantee that the scaled plaintexts are still integers. This means we cannot choose α at log-uniformly over a single log period, since then α only has support on $\{1, 2\}$. Instead, we need to choose α log-uniformly over a sufficiently large multiple of the period that α approximates the continuous log-uniform distribution sufficiently well.
- Second, unlike the shift case, sampling at random from D and then scaling is not the same as sampling from a scaled version of D, since the rounding step does not commute with scaling. For example, for concreteness consider the normal distribution. If we sample from a normal distribution (and round) and then scale, the resulting plaintexts will all be multiples of α. However, if we sample directly from a scaled normal distribution (and then round), the support of the distribution will include integers which are not multiples of α.

  To remedy this issue, we observe that if the plaintexts are sampled from a wide enough distribution, their differing bits will not be amongst the lowest significant bits. Hence, the leakage will actually be independent of the lower order bits. For example, this means that while the rounding does not commute with the scaling, the leakage actually does not depend on the order in which the two operations are carried out.
- The above arguments can be made to work for, say, the normal distribution. However, we would like to have a proof that works for any distribution. Unfortunately, for distributions that oscillate rapidly, we may run into trouble with the above arguments, since rounding such distributions can cause odd behaviors at all scales. This problem is actually unavoidable, as quickly oscillating distributions may have actually have low min-entropy even at large scales. Therefore, we must restrict to "smooth" functions that have a bounded derivative.

  Using a careful analysis, we are able to show for smooth distributions that we achieve the desired scale hiding.

– Finally, we want to have a scheme that is both scale and shift hiding. This is slightly non-trivial, since once we introduce, say, a random shift, we have modified the leakage of the scheme, and cannot directly appeal to the arguments above to obtain scale hiding as well. Instead, we distill a set of specific requirements on the leakage that will work for both shift hiding and scale hiding. We show that our shift hiding scheme above satisfies the requirements needed in order for us to introduce a random scale and additionally prove scale hiding.

Achieving Smoothed CLWW Leakage. Next we turn to actually constructing ORE with smoothed CLWW leakage. Of course, ideal ORE has better than (smoothed) CLWW leakage, so we can construct such ORE based on multilinear maps. However, we want a construction that uses standard tools.

We therefore provide a new construction of ORE using pairings that achieves smoothed CLWW leakage. We believe this construction is of interest on its own,

as it is achieves the to-date smallest leakage of any non-multilinear-map-based scheme.

CLWW ORE and How to Reduce its Leakage. Our construction builds on the ideas of CLWW, so we first briefly recall the ORE scheme of CLWW. In their (basic) scheme, the encryption key is just a PRF key K. To encrypt a plaintext $x \in \{0,1\}^n$, for each prefix $p_i = x[1,\ldots,i]$, the scheme computes

$$y_i = \mathsf{PRF}_K(p_i) + x_{i+1}$$

where x_{i+1} is the $(i+1)$-st bit of x, and the output of $\mathsf{PRF} \in \{0,1\}^\lambda$ is treated as an integer (we will take λ to be the security parameter). The ORE ciphertext is then $(y_1 \ldots, y_n)$. To compare two ciphertexts $(y_1 \ldots, y_n)$ and $(y'_1 \ldots, y'_n)$, one finds the smallest index i such that $y_i \neq y'_i$, and outputs 1 if $y'_i - y_i = 1$. This naturally reveals the index of the bit where the plaintexts differ.

Our approach to reducing the leakage is to attempt to hide the index i where the plaintexts differ. As a naive attempt at this, first consider what happens if we modify the scheme to simply randomly permute the outputs $(y_1 \ldots, y_n)$ (with a fresh permutation chosen for each encryption). We can still compare ciphertexts by appropriately modifying the comparison algorithm: now given $c = (y_1 \ldots, y_n)$ and $c' = (y'_1 \ldots, y'_n)$ (permuted as above), it will look for indices i, j such that either $y'_i - y_j = 1$, in which case it outputs 1, or $y_j - y'_i = 1$, in which case it outputs 0. (If we choose the output length of the PRF to be long enough then this check will be correct with overwhelming probability).

This modification, however, does not actually reduce leakage: an adversary can still determine the most significant differing bit by counting how many elements c and c' have in common.

We can however recover this approach by preventing an adversary from detecting how many elements c and c' have in common. To do so, we introduce and employ the new notion of *property-preserving hashing* (PPH). Intuitively, a PPH is a randomized hashing scheme that is designed to publicly reveal a particular predicate P on pairs of inputs.

PPH can be seen as the hashing (meaning, no decryption) analogue of the notion of property-preserving encryption, a generalization of order-revealing encryption to arbitrary properties due to Pandey and Rouselakis [35]. (This can also be seen as a symmetric-key version of the notion of "relational hash" due to Mandal and Roy [31].)

Specifically, we construct and employ a PPH for the property

$$P_1(x, x') = \begin{cases} 1 & \text{if } x = x' + 1 \\ 0 & \text{otherwise} \end{cases}$$

(Here x, x' are not plaintexts of the ORE scheme, think of them as other inputs determined below.) Security requires that this is *all* that is leaked; in particular, input equality is *not* leaked by the hash values (which requires a randomized hashing algorithm).

Now, the idea is to modify the scheme to include a key K_H for such a PPH $\mathcal{H}$, and the encryption algorithm to not only randomly permute the y_i's but hash them as well, i.e., output $(h_1, \ldots, h_n)$ where $h_i = \mathcal{H}_{K_H}(y_i)$ for the permuted y_i's.[3] The comparison algorithm can again be modified appropriately, namely to not to check if $y'_i - y_j = 1$ but rather if their h'_i and h'_j hash values satisfy P_1 via the PPH (and similarly for the check $y_j - y'_i = 1$).

For any two messages, the resulting ORE scheme is actually ideal: it only reveals the order of the underlying plaintexts, but nothing else. However, for three messages m, m', m'' we see that some additional information is leaked. Namely, if we find that $y'_i - y_j = 1$ $y''_k - y_j = 1$, then we know that $y'_j = y''_k$. We choose the range of the PRF large enough so that this can only happen if y'_j and y'_k are both $\mathsf{PRF}_K(p_\ell) + x_{\ell+1}$ for the same prefix p_ℓ and same bit $x_{\ell+1}$, and y'_j corresponds to the most significant bit where m' differs from m, y''_k corresponds to the most significant bit where m'' differs from m, and moreover these positions are the same. Therefore, the adversary learns whether these most-significant differing bits are the same. It is straightforward to show that this leakage is exactly equivalent to the smoothed CLWW leakage we need. Proving this ORE scheme secure wrt. this leakage based on an achievable notion of security for the PPH turns out to be technically challenging. Nevertheless, we manage to prove it "non-adaptively secure," meaning the adversary is required to non-adaptively choose the dataset, which is realistic for a passive adversary in the outsourced database setting.

Property-Preserving Hash From Bilinear Maps. Next we turn to constructing a property-preserving hash (PPH) for the property $P_1(x, x') = x = x' + 1$. For this, we adapt techniques from perfectly one-way hash functions [9,31] to the symmetric-key setting and use asymmetric bilinear groups. Roughly, in our construction the key for the hash function is a key K for a pseudorandom function PRF and, letting $e\colon G_1 \times G_2 \to G_T$ be an asymmetric bilinear map on prime order cyclic groups G_1, G_2 with generators g_1, g_2, the hash of x is

$$\mathcal{H}_K(x) = (g_1^{r_1}, g_1^{r_1 \mathsf{PRF}_K(x)}, g_2^{r_2}, g_2^{r_2 \mathsf{PRF}_K(x+1)})$$

for fresh random $r_1, r_2 \in \mathbb{Z}_p$. (Thus, the PRF is also pushed to our PPH construction and can be dropped from the higher-level ORE scheme when our hash function is plugged-in). The bilinear map allows testing whether $P_1(x, x')$ from $\mathcal{H}_K(x), \mathcal{H}_K(x')$, and intuitively our use of asymmetric bilinear groups prevents testing other relations such as equality (formally we use the XSDH assumption). We prove the construction secure under an indistinguishability-based notion in which the adversary has to distinguish between the hash of a random challenge x^* and a random hash value, and can query for hash values of inputs x of its

[3] A minor issue here is that we now lose decryptability for the resulting ORE scheme; however, this can easily be added back in a generic way by also encrypting the plaintext separately under a semantically secure scheme.

choice as long as $P_1(x, x^*)$ and $P_1(x^*, x)$ are both 0. Despite being restricted,[4], this notion suffices in our ORE scheme above.

When our PPH is plugged into our ORE scheme, ciphertexts consist of $4n$ group elements, and order comparison requires $n(n-1)$ pairing computations on average. We also note that CLWW gave an improved version of their scheme where ciphertexts are size $O(n)$ rather than $O(n\lambda)$ for security parameter λ, however, we have reason to believe this may be difficult for schemes with our improved leakage profile, see below.

Piecing everything together, we obtain a parameter-hiding ORE from bilinear maps. We note that, as parameter-hiding OPE is impossible, we achieve the first construction of ORE without multilinear maps secure with a security notion that is impossible for OPE.

Generalizing Our ORE Scheme. In our full version [12], we also show several extensions to our smoothed CLWW ORE scheme. In one direction, we achieve an improved level of leakage by considering blocks of bits at a time(encrypting message block by block, rather than bit by bit). We show that if the block size is only 2, then we improve security and efficiency simultaneously, while for larger block sizes the leakage continues to reduce but the efficiency compared to the basic scheme (in terms of both ciphertext size and pairings required for comparison) decreases.

On the other direction, we also show how to improve efficiency while sacrificing some security. We give a more efficient version of the scheme than above (only need $O(n)$ pairings for each comparison), that is still sufficient for achieving parameter-hiding ORE using our conversion.

In addition, we also show how our ORE scheme easily gives a *left/right ORE* as defined by [29] that also improves on their leakage. In left/right ORE, ciphertexts can be generated in either the left mode or right mode, and the comparison algorithm only compares a left and a right ciphertext. Security requires that no information is leaked amongst left and right ciphertexts in isolation.

1.3 Discussion and Perspective

The original OPE scheme of [6] leaks "whatever a random order-preserving function leaks." Unfortunately, this notion does not say anything about what such leakage actually looks like. The situation has been improved in recent works on OPE such as CLWW which define a precise "leakage profile" for their scheme. However, such leakage profiles are still of limited use, since they do not obviously say anything about the actual privacy of the underlying data.

We instead study ORE with a well-defined privacy notion for the underlying plaintexts. A key part of our results is showing how to translate sufficiently strong leakage profiles into such privacy notions. Nonetheless, we do not claim

[4] More generally, following [35] one could allow the adversary to choose two challenge inputs and make queries that do not allow it to trivially distinguish them, but we are unable to prove our construction secure under this stronger notion.

that our new ORE scheme is safe to use in general higher-level protocols. We only claim security as long all that is sensitive is the scale and shift of the underlying plaintext distributions. If, for example, if the shape of the distribution is highly sensitive, or if there are correlations to other data available to the attacker, our notion is insufficient.

However, our construction provably has better leakage than existing efficient schemes, and it at least shows some meaningful security for specific situations. Moreover we suspect that the scheme can be shown to be useful in many other settings by extending our techniques.

1.4 Related Work

Work done on "leaky cryptography" includes work on multiparty computation [33], searchable symmetric and structured encryption [11,13,14,18,21,28,37], and property-preserving encryption [5,6,35]. In the database community, the problem of querying an encrypted database was introduced by Hacigümüş, Iyer, Li and Mehrotra [23], leading to a variety of proposals there but mostly lacking formal security analysis. Proposals of specific outsourced database systems based on property-preserving encryption like ORE include CryptDB [36], Cipherbase [2], and TrustedDB [4].

Besides, in [29], the authors give an efficient ORE construction based on PRFs, while their leakage profile cannot achieve shift hiding and scale hiding simultaneously, which means their scheme cannot meet our privacy notion. Moreover, in [27], the authors give an alternative ORE construction, based on function revealing encryption for simple functions, namely orthogonality testing and intersection cardinality, while their leakage needs further analysis.

2 Background

Notation. All algorithms are assumed to be polynomial-time in the security parameter (though we will sometimes refer to efficient algorithms explicitly). We will denote the security parameter by λ. For a random variable Y, we write $y \xleftarrow{\$} Y$ to denote that y is sampled according to Y's distribution, moreover, let D be Y's distribution, we abuse notation $y \xleftarrow{\$} D$ to mean that y is sampled according to D. For an algorithm A, by $y \xleftarrow{\$} A(x)$ we mean that A is executed on input x and the output is assigned to y, furthermore, if A is randomized, then we write $y \xleftarrow{\$} \mathcal{A}(x)$ to denote running $\mathcal{A}$ on input x with a fresh random tape and letting y be the random variable induced by its output. We denote by $\Pr[A(x) = y : x \xleftarrow{\$} X]$ the probability that A outputs y on input x when x is sampled according to X. We say that an adversary $\mathcal{A}$ has advantage ϵ in distinguishing X from Y if $\Pr[A(x) = 1 : x \xleftarrow{\$} X]$ and $\Pr[A(y) = 1 : y \xleftarrow{\$} Y]$ differ by at most ϵ.

When more convenient, we use the following probability-theoretic notation instead. We write $P_X(x)$ to denote the probability that X places on x, i.e.

$P_X(x) = \Pr[X = x]$, and we say $P_X(x)$ is the probability density function (PDF) of X's distribution. The *statistical distance* between X and Y is given by $\Delta = \frac{1}{2}\sum_x |P_X(x) - P_Y(x)|$. If $\Delta(X,Y)$ is at most ϵ then we say X, Y are ϵ-close. It is well-known that if X, Y are ϵ-close then any (even computationally unbounded) adversary A has advantage at most ϵ in distinguishing X from Y.

The *min-entropy* of a random variable X is $H_\infty(X) = -\log(\max_x P_X(x))$. A value $\nu \in \mathbb{R}$ depending on λ is called *negligible* if its absolute value goes to 0 faster than any polynomial in λ, i.e. $\forall c > 0\ \exists \lambda^* \in \mathbb{N}\ \forall \lambda \geq \lambda^* : |\nu| \leq \frac{1}{\lambda^c}$. We let $[M] = \{1, \ldots, M\}$, $[M]' = \{0, \ldots, M-1\}$ and $[M, N] = \{M, \ldots, N\}$. We write $\boldsymbol{m}$ as a vector of plaintexts and $|\boldsymbol{m}|$ as the vector's length, namely $\boldsymbol{m} = (m_1, \ldots, m_s)$ and $|\boldsymbol{m}| = s$. For a vector $\boldsymbol{m}$, by $a\boldsymbol{m}$ we mean $(am_1, \ldots, am_s)$ and we write $\boldsymbol{m}+b$ to denote $(m_1+b, \ldots, m_s+b)$. Let x be a real number, we write $\lfloor x \rfloor$ as the largest integer s.t. $\lfloor x \rfloor \leq x$, and $\lceil x \rceil$ as the smallest integer s.t. $\lceil x \rceil \geq x$. By $\lfloor x \rceil$, we mean rounding x to the nearest integer, namely $-1/2 \leq \lfloor x \rceil - x < 1/2$. If P is a predicate, we write $\mathbf{1}(P)$ for the function that takes the inputs to P and returns 1 if P holds and 0 otherwise.

PRFs. We use the standard notion of a PRF. A function $F : \{0,1\}^\lambda \times D \to \{0,1\}^\lambda$ is said to be a *PRF with domain D* if for all efficient $\mathcal{A}$ we have that

$$|\Pr[\mathcal{A}^{F(K,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{g(\cdot)}(1^\lambda) = 1]|$$

is a negligible function of λ, where K is uniform over $\{0,1\}^\lambda$ and g is uniform over all functions from D to $\{0,1\}^\lambda$.

ORE. The following definition of syntax for order-revealing encryption makes explicit that comparison may use helper information (e.g. a description of a particular group) by incorporating a *comparison key*, denote ck.

Definition 2 (ORE). *A* ORE scheme *is a tuple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{C})$ with the following syntax.*

- *The key generation algorithm $\mathcal{K}$ is randomized, takes inputs $(1^\lambda, M)$, and always emits two outputs $(\mathsf{sk}, \mathsf{ck})$. We refer to the first output sk as the* secret key *and the second output ck as the* comparison key*.*
- *The encryption algorithm $\mathcal{E}$ is randomized, takes inputs (sk, m) where $m \in [M]$, and always emits a single output c, that we refer to as a* ciphertext*.*
- *The comparison algorithm $\mathcal{C}$ is deterministic, takes inputs (ck, c_1, c_2), and always emits a bit.*

If the comparison algorithm $\mathcal{C}$ is simple integer comparison (i.e., if $\mathcal{C}(\mathsf{ck}, c_1, c_2)$ is a canonical algorithm that treats its the ciphertexts and binary representations of integers and tests which is greater) then the scheme is said to be an order-preserving encryption (OPE) *scheme.*

Correctness of ORE schemes. Intuitively, an ORE scheme is correct if the comparison algorithm can output the order of the underlying plaintext, by taking ck and two ciphertexts as inputs.

Our constructions will only be *computationally* correct, i.e. correct with overwhelming probability when the input messages are provided by an efficient process, under hardness assumptions. Formally, we define correctness using the game $\mathrm{COR}^{\mathsf{ore}}_{\Pi}(\mathcal{A})$, which is defined as follows: The game starts by running $(\mathsf{sk}, \mathsf{ck}) \xleftarrow{\$} \mathcal{K}(1^\lambda, M)$, and it gives ck to $\mathcal{A}$. The adversary $\mathcal{A}$ then outputs two messages $x, y \in [M]$. The game computes $c_1 \xleftarrow{\$} \mathcal{E}(\mathsf{sk}, x)$ and $c_2 \xleftarrow{\$} \mathcal{E}(\mathsf{sk}, y)$, outputs 1 if $x < y$ but $\mathcal{C}(\mathsf{ck}, c_1, c_2) = 0$.

We say that an ORE scheme Π is *computationally correct* if for all efficient adversaries $\mathcal{A}$, all $M = \mathrm{poly}(\lambda)$, we have that $\Pr[\mathrm{COR}^{\mathsf{ore}}_{P_i}(\mathcal{A}) = 1]$ is a negligible function in the security parameter.

Security of ORE Schemes. The following simulation-based security definition is due to Chenette et al. [15]. Here a *leakage profile* is any randomized algorithm. The definition refers to games given in Fig. 3, which we review now. In the real game, key generation is run and the adversary is given the comparison key and oracle access to the encryption algorithm with the corresponding secret key. The adversary eventually outputs a bit that the game uses as its own output. In the ideal simulation game, the adversary is interacting with the same oracle, but the comparison key is generated by a stateful simulator, and the oracle responses are generated by the simulator which receives leakage from the stateful leakage algorithm $\mathcal{L}$.

Game $\mathrm{REAL}^{\mathsf{ore}}_{\Pi}(\mathcal{A})$:	Game $\mathrm{SIM}^{\mathsf{ore}}_{\Pi,\mathcal{L}}(\mathcal{A},\mathcal{S})$:
$(\mathsf{sk}, \mathsf{ck}) \xleftarrow{\$} \mathcal{K}(1^\lambda, M)$; $b \xleftarrow{\$} \mathcal{A}^{\mathrm{Enc}}(\mathsf{ck})$	$\mathsf{st}_\ell \leftarrow \perp$; $(\mathsf{ck}, \mathsf{st}_s) \xleftarrow{\$} S(1^\lambda)$; $b \xleftarrow{\$} \mathcal{A}^{\mathrm{Enc}}(\mathsf{ck})$
Return b	Return b
$\mathrm{Enc}(m)$:	$\mathrm{Enc}(m)$:
Return $\mathcal{E}(\mathsf{sk}, m)$	$(L, \mathsf{st}_\ell) \xleftarrow{\$} \mathcal{L}(\mathsf{st}_\ell, m)$; $(c, \mathsf{st}_s) \xleftarrow{\$} \mathcal{S}(L, \mathsf{st}_s)$
	Return c

Fig. 3. Games $\mathrm{REAL}^{\mathsf{ore}}\Pi(\mathcal{A})$ (left) and $\mathrm{SIM}^{\mathsf{ore}}_{\Pi,\mathcal{L}}(\mathcal{A},\mathcal{S})$ (right), where $\Pi = (\mathcal{E}, \mathcal{C})$ is an ORE scheme, $\mathcal{L}$ is a leakage profile, $\mathcal{A}$ is an adversary, and $\mathcal{S}$ is a simulator.

Definition 3 ($\mathcal{L}$-simulation-security for ORE). *For an ORE scheme Π, an adversary $\mathcal{A}$, a simulator $\mathcal{S}$, and leakage profile $\mathcal{L}$, we define the games* $\mathrm{REAL}^{\mathsf{ore}}_{\Pi}(\mathcal{A})$ *and* $\mathrm{SIM}^{\mathsf{ore}}_{\Pi,\mathcal{L}}(\mathcal{A})$ *in Fig. 3. The advantage of $\mathcal{A}$ with respect to $\mathcal{S}$ is defined as*

$$\mathsf{Adv}^{\mathsf{ore}}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda) = \left|\Pr[\mathrm{REAL}^{\mathsf{ore}}_{\Pi}(\mathcal{A}) = 1] - \Pr[\mathrm{SIM}^{\mathsf{ore}}_{\Pi,\mathcal{L}}(\mathcal{A},\mathcal{S}) = 1]\right|.$$

We say that Π is $\mathcal{L}$-simulation-secure if for every efficient adversary $\mathcal{A}$ there exists an efficient simulator $\mathcal{S}$ such that $\mathsf{Adv}^{\mathsf{ore}}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$ is a negligible function.

We also define non-adaptive *variants of the games where $\mathcal{A}$ gets a single query to an oracle that accepts a vector of messages of unbounded size. In the real*

game $\mathrm{REAL}_{\Pi}^{\mathsf{ore\text{-}na}}(\mathcal{A})$*, the oracle returns the encryptions applied independently to each message. In the ideal game* $\mathrm{SIM}_{\Pi}^{\mathsf{ore\text{-}na}}(\mathcal{A})$*, the leakage function gets the entire vector of messages as input and produces an output* L *that is then given to* $\mathcal{S}$ *which produces a vector of ciphertexts, which are returned by the oracle.*

We define the non-adaptive advantage *of* $\mathcal{A}$ *with respect to* $\mathcal{S}$ *analogously, and denote it* $\mathsf{Adv}^{\mathsf{ore\text{-}na}}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$. Non-adaptive $\mathcal{L}$*-simulation security is defined analogously.*

Ideal ORE. Ideal ORE is the case where the leakage profile $\mathcal{L}$ is simply the list of results of comparisons between the plaintexts. We note that such a $\mathcal{L}$ is *always* revealed by the comparison algorithm, so ideal ORE is the best one can hope for. Ideal ORE can be constructed from multilinear maps [8].

CLWW Leakage. As an example of a non-ideal leakage profile, consider the leakage $\mathcal{L}_{\mathsf{clww}}$ of Chenette, Lewi, Weis and Wu [15]. For $m_0, m_1 \in \{0,1\}^n$, we define the most significant differing bit of m_1 and m_2, denoted $\mathsf{msdb}(m_0, m_1)$, as the index of first bit where m_0, m_1 differ, or $n+1$ if $m_1 = m_2$.

The CLWW leakage profile $\mathcal{L}_{\mathsf{clww}}$ takes in input a vector of plaintext $\boldsymbol{m} = (m_1, \ldots, m_q)$ and produce the following:

$$\mathcal{L}_{\mathsf{clww}}(m_1, \ldots, m_q) := (\forall 1 \le i, j \le n, \mathbf{1}(m_i < m_j), \mathsf{msdb}(m_i, m_j))$$

3 New Security Notions for ORE

In this section, we propose four meaningful notions of privacy: *distribution-hiding*, *parameter-hiding*, *scale-hiding* and *shift-hiding*; in those notions, we are considering the privacy of the underlying *distribution* of data records, rather than the individual data records, and show how to protect information about the underlying data distribution.

Distribution-Hiding for ORE. We assume that all database entries are independently and identically distributed according to some distribution D[5], and the notion of distribution-hiding refers to game defined in Fig. 4. In the interactive game, after receiving the public parameter and comparison key, adversary $\mathcal{A}$ picks two distributions D_0, D_1 and sends to challenger $\mathcal{C}$, $\mathcal{C}$ then flips a coin b, samples a sequence of entries from D_b, and sends back the encrypted entries. Eventually $\mathcal{A}$ outputs a bit, and we say adversary wins if it guesses b correctly. We note that if either of D_b has low min-entropy, it is possible for an adversary to estimate the min-entropy by looking for collisions in its ciphertexts. Therefore, we must restrict D_b to have high min-entropy.

Definition 4 (Distribution-Hiding for ORE). *For an ORE scheme* Π*, an adversary* $\mathcal{A}$*, function* $q = q(\lambda)$ *we define the games* $\mathsf{DH}_{\Pi,q}(\mathcal{A}, \lambda)$ *in Fig. 4. The*

[5] By D, here we mean a sampling algorithm, such that the outputs of this algorithm obey the distribution D, for ease we denote $\max D$ as the maximum item in D's support.

Game $\mathsf{DH}_{\Pi,q}(\mathcal{A},\lambda)$:

$(\mathsf{sk},\mathsf{ck}) \xleftarrow{\$} \mathcal{K}(1^\lambda, M)$; $D_0, D_1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{ck})$ s.t. $H_\infty(D_b) \geq \omega(\log\lambda)$

$b \xleftarrow{\$} \{0,1\}, \boldsymbol{m} \xleftarrow{\$} D_b$ *s.t.* $|m| = q; \max D_b \leq M; \boldsymbol{c} = \mathcal{E}(\mathsf{sk}, \boldsymbol{m})$

$b' = \mathcal{A}(\mathsf{ck}, \boldsymbol{c})$; Return $(b \stackrel{?}{=} b')$

Fig. 4. Games $\mathsf{DH}_{\Pi,q}(\mathcal{A},\lambda)$, where $\Pi = (\mathcal{K},\mathcal{E},\mathcal{C})$ is an ORE scheme, $q = \mathsf{poly}(\lambda)$, and $\mathcal{A}$ is an adversary.

advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}^{\mathsf{DH}}_{\Pi,q}(\mathcal{A},\lambda) = |\Pr[\mathsf{DH}_{\Pi,q}(\mathcal{A},\lambda) - \frac{1}{2}]|$. *We say that Π is distribution-hiding if for every efficient adversary $\mathcal{A}$, and any polynomial $q = \mathsf{poly}(\lambda)$,* $\mathbf{Adv}^{\mathsf{DH}}_{\Pi,q}(\mathcal{A},\lambda)$ *is a negligible function.*

We immediately observe that ideal ORE achieves distribution hiding, while for other known leakier ORE schemes, it's seems unfeasible to achieve this privacy guarantee. However, in many settings, the general shape of the distribution is often known (that is, if the distribution is normal, uniform, Laplace, etc.), and it is reasonable to allow the overall shape to be reveal but hide its mean and/or variance completely, subject to certain restrictions. Before formalize these notion, we firstly introduce some notations.

For a continuous random variable X, where D is X's distribution, we abuse notation $p_D(x) = p_X(x)$. Now we introduce three alternative distributions: $D^\delta_{\mathsf{scale}}, D^\ell_{\mathsf{shift}}, D^{\delta,\ell}_{\mathsf{aff}}$ with parameter δ, ℓ, where the corresponding probability density function is defined as:

$$p_{D_{\mathsf{scale}}} = \frac{p_D(\frac{x}{\delta})}{\delta};\ p_{D_{\mathsf{shift}}}(x) = p_D(x-\ell);\ p_{D_{\mathsf{aff}}} = \frac{p_D(\frac{x-\ell}{\delta})}{\delta}$$

In other words, $D^\delta_{\mathsf{scale}}$ scales the shape of D by a factor of δ; D_{shift} shifts D by ℓ and D_{aff} does both.

Rounded distribution. As our plaintexts are integers, we need map real number to its rounded integer, namely $x \to \lfloor x \rceil$. More precisely, let D be a distribution over real numbers between α and β; we induce a rounded distribution $R^{\alpha,\beta}_D$ on $[\lceil\alpha\rceil, \lfloor\beta\rfloor]$which samples from D and then rounds. Its probability density function is:

$$p_{R^{\alpha,\beta}_D}(k) = \begin{cases} \frac{\int_\alpha^{\lceil\alpha\rceil+1/2} p_D(x)dx}{\int_\alpha^\beta p_D(x)dx} & k = \alpha \\ \frac{\int_{k-1/2}^{k+1/2} p_D(x)dx}{\int_\alpha^\beta p_D(x)dx} & k \in [\lceil\alpha+1\rceil, \lfloor\beta-1\rfloor] \\ \frac{\int_{\lfloor\beta\rfloor-1/2}^{\beta} p_D(x)dx}{\int_\alpha^\beta p_D(x)dx} & k = \beta \\ 0 & \textit{Otherwise} \end{cases}$$

In the case of $D^\delta_{\mathsf{scale}}$, D^ℓ_{shift}, or $D^{\delta,\ell}_{\mathsf{aff}}$, we will use the notation $\lfloor D^\delta_{\mathsf{scale}} \rceil$, $\lfloor D^\ell_{\mathsf{shift}} \rceil$, and $\lfloor D^{\delta,\ell}_{\mathsf{aff}} \rceil$ to denote the respective rounded distributions.

Now, we present the notion "(γ, D)-parameter-hiding" ORE, referring to the game defined in Fig. 5. Here, D is a distribution over $[0, 1]$, which represents the description of the known shape of the distribution of plaintexts. γ is a lower-bound on the scaling that is allowed. Then key generation is run and adversary is given the public parameter, (γ, D), and the comparison key. Then, the adversary $\mathcal{A}$ sends two pairs of parameters $(\delta_0, \ell_0), (\delta_1, \ell_1)$ to challenger $\mathcal{C}$. Next, $\mathcal{C}$ flips a coin b, checks whether the parameter is proper($\mathbf{1}(\delta_0 \geq \gamma \cap \delta_1 \geq \gamma)$), then samples a sequence of data entries from the rounded distribution $\lfloor D_{\mathsf{aff}}^{\delta_b, \ell_b} \rceil$ and sends back encrypted data. Eventually $\mathcal{A}$ outputs a bit, and we say adversary wins if it guesses b correctly.

Game (γ, D)-$\mathsf{para\text{-}hid}_{\Pi,q}(\mathcal{A}, \lambda)$:

$(\mathsf{sk}, \mathsf{ck}) \xleftarrow{\$} \mathcal{K}(1^\lambda, M)$; $\delta_0, \ell_0, \delta_1, \ell_1 \leftarrow \mathcal{A}(\mathsf{ck}, D)$

If $\delta_0 < \gamma$ or $\delta_1 < \gamma$, output a random bit and abort,

else, $b \xleftarrow{\$} \{0, 1\}, \boldsymbol{m} \xleftarrow{\$} \lfloor D_{\mathsf{aff}}^{\delta_b, \ell_b} \rceil, s.t. |m| = q; \max \lfloor D_{\mathsf{aff}}^{\delta_b, \ell_b} \rceil \leq M; \boldsymbol{c} = \mathcal{E}(\mathsf{sk}, \boldsymbol{m})$

$b' = \mathcal{A}(\mathsf{ck}, \boldsymbol{c})$ Return $(b \stackrel{?}{=} b')$

Fig. 5. Games $\mathsf{para\text{-}hid}_{\Pi,q}(\mathcal{A}, \lambda)$, where $\Pi = (\mathcal{E}, \mathcal{C})$ is an ORE scheme, D is a distribution on $[0, 1]$, $\mathcal{A}$ is an adversary

Definition 5 ((γ, D)-parameter hiding for ORE). *For an ORE scheme Π, an adversary $\mathcal{A}$, a distribution D, and function $q = q(\lambda)$, we define the games (γ, D)-$\mathsf{para\text{-}hid}_{\Pi,q}(\mathcal{A}, \lambda)$ in Fig. 5. The advantage of $\mathcal{A}$ is defined as*

$$\mathbf{Adv}^{\mathsf{para\text{-}hid}}_{\Pi,q,\gamma,D}(\mathcal{A}, \lambda) = |\Pr[(\gamma, D)\text{-}\mathsf{para\text{-}hid}_{\Pi,q}(\mathcal{A}, \lambda) - \frac{1}{2}]|$$

We say that Π is (γ, D)-parameter hiding if for every efficient adversary $\mathcal{A}$ and polynomial q $\mathbf{Adv}^{\mathsf{para\text{-}hid}}_{\Pi,q,\gamma,D}(\mathcal{A}, \lambda)$ is a negligible function.

Similarly, we define (γ, D)-scale hiding and (γ, D)-shift hiding with little change as above. More precisely, in the game of (γ, D)-scale hiding, we add the restriction $\ell_0 = \ell_1 = 0$ and in the game of (γ, D)-shift hiding, we add the restriction $\delta_0 = \delta_1$. Due to the space limit, we skip the formal definitions here.

We note that these three notions are distribution dependent, and we would like they work for any distribution. Unfortunately, quickly oscillating distributions do not fit into our case, as they may have actually low min-entropy for their discretized distributions on integers, even at large scales. Hence, we place additional restrictions. We place the following restriction, which is sufficient, but potentially stronger than necessary:

(η, μ)-SMOOTH DISTRIBUTION. We let D be a distribution where its support mainly on $[0, 1]$ ($\Pr[x \notin [0, 1] : x \leftarrow D] \leq \mathsf{negl}(\lambda)$), we denote $p'_D(x)$ as its derivative, and we say that D is (η, μ)-smooth if (1) $\forall x \in [0, 1], p_D(x) \leq \eta$; (2) $|p'_D(x)| \leq \eta$ for all $x \in [0, 1]$ except for μ points.

Definition 6 ((γ, η, μ)-parameter hiding for ORE). *For an ORE scheme Π, we say Π is (γ, η, μ)-parameter hiding if for every efficient adversary $\mathcal{A}$, polynomial q, and any (η, μ)-smooth distribution D, $\mathbf{Adv}^{\mathsf{para\text{-}hid}}_{\Pi,q,\gamma,D}(\mathcal{A}, \lambda)$ is a negligible function.*

4 Parameter Hiding ORE

In this section, we will assume we are given an ORE $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{C})$ with a "smoothed" version of CLWW leakage, defined below. Later, in Sect. 5, we will show how to instantiate such a scheme from bilinear maps.

We show how to convert a scheme with smoothed CLWW leakage into a parameter-hiding ORE scheme by simply composing with a linear function: namely, for any plaintext m, the ciphertext has form $\mathcal{E}(\alpha m + \beta)$, where α, β are the same across all messages and are sampled as part of the secret key. Intuitively, α helps to hide the scale parameter and β hides the shift. We need to be careful about the distributions of α and β; α needs to be drawn from a "discrete log uniform" distribution of appropriate domain, and β needs to be chosen from a uniform distribution of appropriate domain.

The discrete log uniform distribution D on $[A, B]$ ($\mathsf{logU}(A, B)$) has probability density function:

$$p_D(k) = \begin{cases} \frac{1/k}{\sum_{i=A}^{B} 1/i} & i \in [A, B] \\ 0 & \textit{Otherwise} \end{cases}$$

We say a leakage function $\mathcal{L}$ is smoothed CLWW if:

1. For any two plaintext sequences $\boldsymbol{m}_0, \boldsymbol{m_1}$, if $\mathcal{L}_{\mathsf{clww}}(\boldsymbol{m}_0) = \mathcal{L}_{\mathsf{clww}}(\boldsymbol{m}_1)$, then $\mathcal{L}(\boldsymbol{m}_0) = \mathcal{L}(\boldsymbol{m}_1)$ (in other words, it leaks no more information that CLWW);
2. For any plaintext sequence $\boldsymbol{m}$, $\mathcal{L}(\boldsymbol{m}) = \mathcal{L}(2\boldsymbol{m})$

4.1 Parameter-Hiding ORE

In this part, we give the formal description of parameter-hiding ORE. To simplify our exposition, we first specify some parameters. We will assume we are given:

$$q = \mathsf{poly}(\lambda), M = 2^{\mathsf{poly}(\lambda)}, \gamma = 2^{\omega(\log \lambda)}, \eta, \mu \leq O(1)$$

We will assume γ and M are exactly powers of 2 without loss of generality by rounding up. We define:

$$\tau = \gamma, \xi = \gamma^2, U = 4\xi M, T = \gamma^2 \times U, K = 2 \times T$$

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{C})$ be an ORE scheme on message space $[K]$ with smoothed CLWW leakage $\mathcal{L}$. We define our new ORE $\Pi_{\mathsf{aff}} = (\mathcal{K}_{\mathsf{aff}}, \mathcal{E}_{\mathsf{aff}}, \mathcal{C}_{\mathsf{aff}})$ on message space $[M]$ as follows:

- $\mathcal{K}_{\mathsf{aff}}(1^\lambda, M, \Pi)$: On input the security parameter λ, message space $[M]$ and Π, the algorithm picks a super-polynomial $\gamma = 2^{\omega(\log \lambda)}$ as a global parameter, and computes parameters above. Then it runs $(\mathsf{ck}, \mathsf{sk}) \leftarrow \mathcal{K}(1^\lambda, K)$, draws $\alpha \xleftarrow{\$} \mathsf{logU}(\xi, 2\xi - 1)$ and β from discrete uniform on $[T]'$ and outputs $\mathsf{sk}_{\mathsf{aff}} = (\mathsf{sk}, \alpha, \beta), \mathsf{ck}_{\mathsf{aff}} = \mathsf{ck}$;
- $\mathcal{E}_{\mathsf{aff}}(\mathsf{sk}_{\mathsf{aff}}, m)$. On input the secret key $\mathsf{sk}_{\mathsf{aff}}$ and a message $m \in [M]$, it outputs

$$\mathsf{CT}_{\mathsf{aff}} = \mathcal{E}(\alpha m + \beta)$$

  By our choice of message space $[K]$ for Π, the input to $\mathcal{E}$ is guaranteed to be in the message space.
- $\mathcal{C}_{\mathsf{aff}}(\mathsf{ck}_{\mathsf{aff}}, \mathsf{CT}^0_{\mathsf{aff}}, \mathsf{CT}^1_{\mathsf{aff}})$: On inputs the comparison key $\mathsf{ck}_{\mathsf{aff}}$, two ciphertexts $\mathsf{CT}^0_{\mathsf{aff}}, \mathsf{CT}^1_{\mathsf{aff}}$, it outputs $\mathcal{C}(\mathsf{ck}_{\mathsf{aff}}, \mathsf{CT}^0_{\mathsf{aff}}, \mathsf{CT}^1_{\mathsf{aff}})$

Here we also give the description of composted schemes that only achieve "scale-hiding" or "shift-hiding". Formally, we define $\Pi_{\mathsf{scale}} = (\mathcal{K}_{\mathsf{scale}}, \mathcal{E}_{\mathsf{scale}}, \mathcal{C}_{\mathsf{scale}})$ and $\Pi_{\mathsf{shift}} = (\mathcal{K}_{\mathsf{shift}}, \mathcal{E}_{\mathsf{shift}}, \mathcal{C}_{\mathsf{shift}})$, respectively:

- $\mathcal{K}_{\mathsf{scale}}(1^\lambda, M, \Pi)$: On input the security parameter λ, the message space $[M]$ and Π, the algorithm picks a super-polynomial $\gamma = 2^{\omega(\log \lambda)}$ as a global parameter, and computes parameters above. Then it runs $(\mathsf{ck}, \mathsf{sk}) \leftarrow \mathcal{K}(1^\lambda, K)$, draws $\alpha \xleftarrow{\$} \mathsf{logU}(\xi, 2\xi - 1)$ and outputs $\mathsf{sk}_{\mathsf{scale}} = (\mathsf{sk}, \alpha), \mathsf{ck}_{\mathsf{scale}} = \mathsf{ck}$;
- $\mathcal{E}_{\mathsf{scale}}(\mathsf{sk}_{\mathsf{scale}}, m)$. On input the secret key $\mathsf{sk}_{\mathsf{scale}}$ and a message $m \in [M]$, it outputs

$$\mathsf{CT}_{\mathsf{scale}} = \mathcal{E}(\alpha m)$$

- $\mathcal{C}_{\mathsf{scale}}(\mathsf{ck}_{\mathsf{scale}}, \mathsf{CT}^0_{\mathsf{scale}}, \mathsf{CT}^1_{\mathsf{scale}})$: On inputs the comparison key $\mathsf{ck}_{\mathsf{scale}}$, two ciphertexts $\mathsf{CT}^0_{\mathsf{scale}}, \mathsf{CT}^1_{\mathsf{scale}}$, it outputs $\mathcal{C}(\mathsf{ck}_{\mathsf{scale}}, \mathsf{CT}^0_{\mathsf{scale}}, \mathsf{CT}^1_{\mathsf{scale}})$.
- $\mathcal{K}_{\mathsf{shift}}(1^\lambda, M, \Pi)$: On input the security parameter λ, the message space $[M]$ and Π, the algorithm picks a super-polynomial $\gamma = 2^{\omega(\log \lambda)}$ as a global parameter, and computes parameters above. Then it runs $(\mathsf{ck}, \mathsf{sk}) \leftarrow \mathcal{K}(1^\lambda)$, draws β from discrete uniform on $[T]'$ and outputs $\mathsf{sk}_{\mathsf{shift}} = (\mathsf{sk}, \alpha), \mathsf{ck}_{\mathsf{shift}} = \mathsf{ck}$;
- $\mathcal{E}_{\mathsf{shift}}(\mathsf{sk}_{\mathsf{shift}}, m)$. On input the secret key $\mathsf{sk}_{\mathsf{shift}}$ and a message $m \in [M]$, it outputs

$$\mathsf{CT}_{\mathsf{shift}} = \mathcal{E}(m + b)$$

- $\mathcal{C}_{\mathsf{shift}}(\mathsf{ck}_{\mathsf{shift}}, \mathsf{CT}^0_{\mathsf{shift}}, \mathsf{CT}^1_{\mathsf{shift}})$: On inputs the comparison key $\mathsf{ck}_{\mathsf{shift}}$, two ciphertexts $\mathsf{CT}^0_{\mathsf{shift}}, \mathsf{CT}^1_{\mathsf{shift}}$, it outputs $\mathcal{C}(\mathsf{ck}_{\mathsf{shift}}, \mathsf{CT}^0_{\mathsf{shift}}, \mathsf{CT}^1_{\mathsf{shift}})$.

The correctness of $\Pi_{\mathsf{aff}}, \Pi_{\mathsf{scale}}$ and Π_{shift} is directly held by correctness of Π, and what is more interesting is the privacy that those scheme can guarantee.

4.2 Main Theorem

In the part, we prove Π_{aff} is parameter hiding, formally:

Theorem 7 (Main Theorem). *Assuming Π has $\mathcal{L}$-simulation-security where $\mathcal{L}$ is smoothed CLWW, then for any $\gamma = 2^{\omega(\log \lambda)}$, Π_{aff} is (γ, η, μ)-parameter hiding.*

Proof. According to the security notions, it is straightforward that if an ORE scheme is (γ, η, μ)-parameter hiding, then it is also (γ, η, μ)-scale hiding and (γ, η, μ)-shift hiding. Next we claim the converse proposition holds.

Claim. If an ORE scheme Π achieves (γ, η, μ)-scale hiding and (γ, η, μ)-shift hiding simultaneously, then Π is (γ, η, μ)-parameter hiding.

We sketch the proof by hybrid argument. For any $\gamma = 2^{\omega(\log \lambda)}$ and (η, μ)-smooth distribution D, firstly, by shift-hiding, there is no efficient adversary that distinguish (δ_0, ℓ_0) from $(\delta_0, 0)$ with non-negligible probability. Then due to scale-hiding, no efficient adversary can differ $(\delta_0, 0)$ from $(\delta_1, 0)$ with non-negligible probability. Thirdly, same as the first argument, any efficient adversary can distinguish $(\delta_1, 0)$ from (δ_1, ℓ_1) with only negligible advantage. Combining together, Π achieves (γ, η, μ)-parameter hiding.

Thus, it suffices to show Π_{aff} is both (γ, η, μ)-scale hiding and (γ, η, μ)-shift hiding, due to space limit, we put the rigorous proof in our full version [12].

5 ORE with Smoothed CLWW Leakage

We start by defining the security we target via a smoothed CLWW leakage function. Then we recall a primitive for our construction called a *property-preserving hash (PPH) function*, and state and analyze our ORE construction using a PPH. In a later section we instantiate the PPH to complete the construction. Next, we give variant constructions with trade-offs between efficiency and leakage.

Now We define the non-adaptive version of the leakage profile for our construction. The leakage profile takes in input a vector of messages $\boldsymbol{m} = (m_1, \ldots, m_q)$ and produces the following:

$$\mathcal{L}_f(m_1, \ldots, m_q) := (\forall 1 \leq i, j, k \leq q, \mathbf{1}(m_i < m_j), \mathbf{1}(\mathsf{msdb}(m_i, m_j) = \mathsf{msdb}(m_i, m_k)))$$

By definition, it's easy to note that $\mathcal{L}_f$ leaks strictly less than CLWW. Except for the order of underlying plaintexts, it only leaks whether the position of $\mathsf{msdb}(m_i, m_j)$ and $\mathsf{msdb}(m_i, m_j)$ are the same, therefore the leakage profile preserve consistent if we left-shift all the plaintexts by one bit, which referring to $\mathcal{L}_f(\boldsymbol{m}) = \mathcal{L}_f(2\boldsymbol{m})$. Thus, $\mathcal{L}_f$ is smoothed CLWW.

5.1 Property Preserving Hash

Our construction will depend on a tool – *property preserving hash (PPH)*, which is essentially a property-preserving encryption scheme [35] without the decryption algorithm. In this section we recall the syntax and security of a PPH.

Definition 8. *A* property-preserving hash (PPH) scheme *is a tuple of algorithms* $\Gamma = (\mathcal{K}_h, \mathcal{H}, \mathcal{T})$ *with the following syntax:*

- *The key generation algorithm* $\mathcal{K}_h$ *is randomized, takes as input* 1^λ *and emits two outputs* $(\mathsf{hk}, \mathsf{tk})$ *that we refer to as the* hash key hk *and* test key tk. *These implicitly define a domain* D *and range* R *for the hash.*
- *The evaluation algorithm* $\mathcal{H}$ *is randomized, takes as input the hash key* hk, *an input* $x \in D$, *and emits a single output* $h \in R$ *that we refer to as the* hash of x.
- *The test algorithm* $\mathcal{T}$ *is deterministic, takes as input the test key* tk *and two hashes* h_1, h_2, *and emits a bit.*

Correctness of PPH schemes. Let P be a predicate on pairs of inputs. We define correctness of a PPH Γ with respect to P via the game $\mathrm{COR}^{\mathsf{pph}}_{\Gamma,P}(\mathcal{A})$, which is as follows: It starts by running $(\mathsf{hk}, \mathsf{tk}) \xleftarrow{\$} \mathcal{K}_h(1^\lambda)$ and gives tk to $\mathcal{A}$. Then $\mathcal{A}$ outputs x, y. The game computes $h \xleftarrow{\$} \mathcal{H}(\mathsf{hk}, x), h' \xleftarrow{\$} \mathcal{H}(\mathsf{hk}, y)$ and outputs 1 if $\mathcal{T}(\mathsf{tk}, h, h') \neq P(x, y)$. We say that Γ is *computationally correct with respect to* P if for all efficient $\mathcal{A}$, $\Pr[\mathrm{COR}^{\mathsf{pph}}_{\Gamma,P}(\mathcal{A}) = 1]$ is a negligible function of λ.

Security of PPH Schemes. We recall a simplified version of the security definition for PPH that is a weaker version of PPE security defined by Pandey and Rouselakis [35]. The definition is a sort of semantic security for random messages under chosen-plaintext attacks, except that the adversary is restricted from making certain queries.

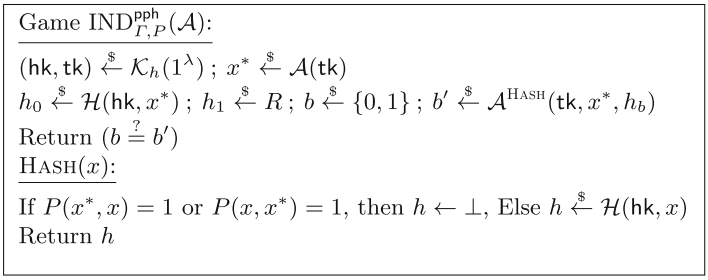

Fig. 6. Game $\mathrm{IND}^{\mathsf{pph}}_{\Gamma,P}(\mathcal{A})$.

Definition 9. *Let* P *be some predicate and* $\Gamma = (\mathcal{K}_h, \mathcal{H}, \mathcal{T})$ *be a PPH scheme with respect to* P. *For an adversary* $\mathcal{A}$ *we define the game* $\mathrm{IND}^{\mathsf{pph}}_{\Gamma,P}(\mathcal{A})$ *in Fig. 6. The* restricted-chosen-input advantage of $\mathcal{A}$ *is defined to be* $\mathsf{Adv}^{\mathsf{pph}}_{\Gamma,P,\mathcal{A}}(\lambda) = 2\Pr[\mathrm{IND}^{\mathsf{pph}}_{\Gamma,P}(\mathcal{A}) = 1] - 1$. *We say that* Γ *is* restricted-chosen-input secure *if for all efficient adversaries* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{pph}}_{\Gamma,P,\mathcal{A}}(\lambda)$ *is negligible.*

5.2 ORE from PPH

Construction. Let $F : K \times ([n] \times \{0,1\}^n) \to \{0,1\}^\lambda$ be a secure PRF. Let $P(x, y) = \mathbf{1}(x = y + 1)$ be the predicate that outputs 1 if and only if $x = y + 1$,

and let $\Gamma = (\mathcal{K}_h, \mathcal{H}, \mathcal{T})$ be a PPH scheme with respect to P. In our construction, we interpret the output of F as a λ-bit integer, which is also the input domain of the PPH Γ. We define our ORE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{C})$ as follows:

- $\mathcal{K}(1^\lambda, M)$: On input the security parameter and message space $[M]$, the algorithm chooses a key k uniformly at random for F, and runs the key generation algorithm of the property preserving hash function $\Gamma.\mathcal{K}_h$ to obtain the hash and test keys $(\mathsf{hk}, \mathsf{tk})$. It sets $\mathsf{ck} \leftarrow \mathsf{tk}$, $\mathsf{sk} \leftarrow (k, \mathsf{hk})$ and outputs $(\mathsf{ck}, \mathsf{sk})$.
- $\mathcal{E}(\mathsf{sk}, m)$: On input the secret key sk and a message m, the algorithm writes the binary representation as m as $(b_1, \ldots, b_n)$, and then for $i = 1, \ldots, n$, it computes:

$$u_i = F(k, (i, b_1 b_2 \cdots b_{i-1} || 0^{n-i+1})) + b_i \mod 2^\lambda, \quad t_i = \Gamma.\mathcal{H}(\mathsf{hk}, u_i).$$

  We note that u_i is computed by treating the PRF output as a member of $\{0, \ldots, 2^\lambda - 1\}$. Then it chooses a random permutation $\pi : [n] \to [n]$, and sets $v_i = t_{\pi(i)}$. The algorithm outputs $\mathsf{CT} = (v_1, \ldots, v_n)$.
- $\mathcal{C}(\mathsf{ck}, \mathsf{CT}_1, \mathsf{CT}_2)$: on input the public parameter, two ciphertexts $\mathsf{CT}_1, \mathsf{CT}_2$ where $\mathsf{CT}_1 = (v_1, \ldots, v_n), \mathsf{CT}_2 = (v'_1, \ldots, v'_n)$, the algorithm runs $\Gamma.\mathcal{T}(\mathsf{tk}, v_i, v'_j)$ and $\Gamma.\mathcal{T}(\mathsf{tk}, v'_i, v_j)$ for every $i, j \in [n]$. If there exists a pair (i^*, j^*) such that $\Gamma.\mathcal{T}(\mathsf{tk}, v_{i^*}, v'_{j^*}) = 1$, then the algorithm outputs 1, meaning $m_1 > m_2$; else if there exists a pair (i^*, j^*) such that $\Gamma.\mathcal{T}(\mathsf{tk}, v'_{i^*}, v_{j^*}) = 1$, then the algorithm outputs 0, meaning $m_1 < m_2$; otherwise it outputs $\perp$, meaning $m_1 = m_2$.

Correctness. For two messages m_1, m_2, let $(b_1, \ldots b_n)$ and $(b'_1, \ldots, b'_n)$ be their binary representations. Assuming $m_1 > m_2$, there must exists a unique index $i^* \in [n]$ such that $u_i = u'_i + 1$. Therefore correctness of Π is followed by correctness of PPH. We can use the same argument for the case $m_1 = m_2$ and $m_1 < m_2$. What is more interesting is its simulation based security, as it is the foundation for parameter hiding ORE, formally:

Theorem 10. *Assuming F is a secure PRF and Γ is restricted-chosen-input secure, Π is $\mathcal{L}_{\mathsf{f}}$-non-adaptively-simulation secure.*

Proof. We use a hybrid argument, and define a sequence of hybrid games as follows:

- H_{-1}: Real game $\mathrm{REAL}^{\mathsf{ore}}_{\Pi}(\mathcal{A})$;
- H_0: Same as H_{-1}, except replacing PRF $F_k(\cdot)$ by a truely random function F^* in the encryption oracle;
- $\mathsf{H}_{i \cdot q + j}$ Depend on a predicate $\mathsf{Switch}_{(i,j)}$ which is define below. If $\mathsf{Switch}_{(i,j)} = 0$, then $\mathsf{H}_{i \cdot q + j} = \mathsf{H}_{i \cdot q + j - 1}$, else in procedure of $\mathcal{E}(m_j)$, u^j_i is replaced by a random string.

From the high level, we establish the proof by showing show that any adjacent hybrids are indistinguishable, and then we construct an efficient simulator S such

that the output of H_{qn} and $\mathrm{SIM}^{\mathsf{ore}}_{\Pi,\mathcal{L}_{\mathsf{f}}}(\mathcal{A},\mathcal{S})$ are statistically identical. For the predicate, we say $\mathsf{Switch}_{i,j}=1$ if $\forall k\in[q], \mathsf{msdb}(m_j,m_k)\neq i$, and 0 otherwise. We note that when $\mathsf{Switch}_{i,j}=0$, there exists u_i^k such that $u_i^j=u_i^k\pm 1$, the relation which can be detected by the test algorithm of PPH(for the i-th bit of m_j, we call such a bit a leaky bit), which means we cannot replace it with random string, otherwise adversary can trivially distinguish it. In the following we firstly prove any adjacent objects are computational indistinguishable.

Lemma 11. *Assuming Γ is restricted-chosen-input secure, for any $k\in[qn]$ $\mathsf{H}_{k-1}\overset{\mathsf{comp}}{\approx}\mathsf{H}_k$.*

Proof. Due to the security of PRF, it's trivial that $\mathsf{H}_{-1}\overset{\mathsf{comp}}{\approx}\mathsf{H}_0$, and for any $k>0$ (for ease, $k=i^*\cdot q+j^*$ where $i^*\in[n-1], j^*\in[q]$), it suffices to show $\mathsf{H}_{k-1}\overset{\mathsf{comp}}{\approx}\mathsf{H}_k$ under the condition $\mathsf{Switch}_{i^*,j^*}=1$($\mathsf{Switch}_{i^*,j^*}=0$ implies $\mathsf{H}_{k-1}=\mathsf{H}_k$). We prove that if there exists adversary $\mathcal{A}$ that distinguish $\mathsf{H_k}$ from $\mathsf{H_{k-1}}$ with noticeable advantage ϵ, then we can construct a simulator $\mathcal{B}$ wins the restricted-chosen-input game with ϵ-negl. Here is the description of $\mathcal{B}$. Firstly it runs $\mathrm{IND}^{\mathsf{pph}}_{\Gamma}$, and sends tk as the comparison key ck to $\mathcal{A}$. After receiving a sequence of plaintext $m_1,\ldots,m_q$, it picks a random function F^*(using the lazy sampling technique for instance), sets $X^*=F^*(i^*,b_1^{j^*}b_2^{j^*}\cdots b_{i^*-1}^{j^*}||0^{n-i^*+1})+b_{i^*}^{j^*}$ where b_i^j is the i-th bit of m_j. Then it sends X^* to its challenger in restricted-chosen-input game and gets back T as the challenge term. To simulate the encryption oracle, $\mathcal{B}$ works as follows:

1. $(i',j')>(i^*,j^*)$(here using a natural order for tuples, $(i,j)>(i',j')$ iff $iq+j>i'q+j'$), $\mathcal{B}$ computes:
$$u_{i'}^{j'}=F^*(i^*,b_1^{j'}b_2^{j'}\cdots b_{i'-1}^{j'}||0^{n-i'+1})+b_{i'}^{j'};t_{i'}^{j'}=\Gamma.\mathcal{H}(\mathsf{hk},u_{i'}^{j'})$$
2. $(i',j')<(i^*,j^*)\cap\mathsf{Switch}_{i',j'}=0$, then same as above, else $u_{i'}^{j'}\overset{\$}{\leftarrow}\{0,1\}^\lambda, t_{i'}^{j'}=\Gamma.\mathcal{H}(\mathsf{hk},u_{i'}^{j'})$.
3. sets $t_{i^*}^{j^*}=T$, and $\forall j\in[q]$, picks a random permutation π_j and outputs the ciphertexts $\mathsf{CT}_j=(t^j_{\pi_j(1)},\ldots,t^j_{\pi_j(n)})$.

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs[6].

Since F^* is a random function, $\Pr[u_{i'}^{j'}=X^*\pm 1]$ is negligible for all $(i',j')\neq(i^*,j^*)$, which means $\mathcal{B}$ fails to simulate the encryption oracle with only negligible probability. Besides, when $T=\Gamma.\mathcal{H}(\mathsf{hk},X^*)$, $\mathcal{B}$ properly simulates H_{k-1}, and if T is random, then $\mathcal{B}$ simulates H_k(due to the PRF security, the distribution of $\Gamma.\mathcal{H}(\mathsf{hk},r):r\overset{\$}{\leftarrow}\{0,1\}^\lambda$ is computationally close to a random variable that uniformly sampled from the range of Γ). Hence, if $\mathbf{Adv}(\mathcal{A})$ is noticeable, then $\mathcal{B}$'s advantage is also noticeable. □

In the following, we describe an efficient simulator S such that the output of H_{qn} and $\mathrm{SIM}^{\mathsf{ore}}_{\Pi,\mathcal{L}_{\mathsf{f}}}(\mathcal{A},\mathcal{S})$ are statistically identical. Roughly speaking, we note

[6] We note that $\mathcal{B}$ does not have hk, what it does is to call the hash oracle.

that $\mathsf{Switch}_{i,j} = 1$ means that i-th bit of m_j is not a leaky bit, indicating that its value would not affect the leakage profile whp. Hence, it suffices to only simulate the leaky bit of each individual message, which can be extracted by $\mathcal{L}_f$, and sets the rest just as random string. Due to the final random permutations, H_{qn} and $\mathrm{SIM}^{\mathsf{ore}}_{\Pi,\mathcal{L}_\mathsf{f}}(\mathcal{A},\mathcal{S})$ are statistically identical. Formally:

Description of the simulator. For fixed a message set $\mathcal{M} = \{m_1, \ldots, m_q\}$ (without loss of generality, we assume $m_1 > \ldots > m_q$), the simulator $\mathcal{S}$ is given the leakage information $\mathcal{L}_f(m_1, \ldots, m_q)$. $\mathcal{S}$ firstly keeps a $q \times n$ matrix $\mathcal{B}$ and runs a recursive algorithm $\mathsf{FillMatrix}(1, 1, q)$ to fill in the entries, as follows:

- If $j = k$, then $\forall i' \in [i, n]$, $\mathcal{B}[j][i'] = r$ where $r \xleftarrow{\$} \{0,1\}^\lambda$;
- Else, it proceeds as follows:
  - searches the smallest $j^* \in [j, k]$ s.t. $P(m_j, m_{j^*}) = P(m_j, m_k)$;
  - sets $\mathcal{B}[j'][i] = r', \forall j' \in [j, j^* - 1]; \mathcal{B}[j'][i] = r' - 1, \forall j' \in [j^*, k]$, where $r' \xleftarrow{\$} \{0,1\}^\lambda$;
  - runs $\mathsf{FillMatrix}(i + 1, j, j' - 1)$ and $\mathsf{FillMatrix}(i + 1, j', k)$ recursively.

More concretely, our recursive algorithm is to fill in the entries by

$$\mathsf{FillMatrix}(i, j, k),\ \forall i \in [n], j \le k \in [q]$$

Then $\mathcal{S}$ runs $\Gamma.\mathcal{K}_h(1^\lambda)$ and gets the keys tk, hk, and sets $t_{i,j} = \Gamma.\mathcal{H}(\mathsf{hk}, \mathcal{B}[j][i])$, $\forall i \in [n], j \in [q]$. Finally, $\mathcal{S}$ samples random permutations π_j, outputs CT_j as $\mathsf{CT}_j = (t^j_{\pi_j(1)}, \ldots, t^j_{\pi_j(n)})$ We note that the FillMatrix algorithm terminates after at most qn steps as each cell will not be written twice, hence $\mathcal{S}$ is an efficient simulator.

Finally we claim that $\mathcal{S}$ properly simulates the relevant games. We first observe that the simulator identifies how many leaked bits (prefixes) there are for the messages $m_1, \ldots, m_q$. Recall that if messages $m_1, \ldots, m_q$ share the same prefix up to the $\ell - 1$-th bit, and if there exists (the first) i^* such that $\mathsf{msdb}(m_1, m_{i^*}) = \mathsf{msdb}(m_1, m_q)$, then we can conclude that $\{m_1, \ldots, m_{i^*-1}\}$ has 1 on their ℓ-th bit, and $\{m_{i^*}, \ldots, m_q\}$ has 0 on their ℓ-th bit. This way the ℓ-th bit of these messages are leaked. The simulator recursively identifies other leaked bits for these two sets. At the end, for each message, how many prefixes whose next bits are leaked will be identified. As this information will also be identified in the hybrid H_{qn}. So a random permutation (for H_{qn} and the simulation) will hide these leaked prefixes, except the total number. Thus, our simulation is identical to H_{qn}, and we establish the entire proof. □

5.3 More Efficient Comparisons

The construction above needs to run $O(n^2)$ times PPH test algorithm for one single comparison, which is very expensive for real application. In this part, we present a variant ORE achieving better efficiency but with a weaker leakage profile, which only requires $O(n)$ pairings in each individual comparison. And what's more interesting is that this weaker leakage profile is also smoothed

CLWW, that means we can still construct a parameter hiding ORE based on it, along with better efficiency. From the high level, we fix a permutation for all encryptions(this permutation is part of the secret key now), rather than sampling fresh permutation for each ciphertext. Therefore, in the comparison, we only need run the PPH test for pairs that share the same index, which means only $O(n)$ pairings for one comparison. Formally:

Construction. Let F be a secure PRF with the same syntax as above, let $P(x, y) = \mathbf{1}(x = y + 1)$ be the relation predicate that outputs 1 if and only if $x = y+1$, and let $\Gamma = (\mathcal{K}_h, \mathcal{H}, \mathcal{T})$ be a PPH scheme with respect to P, as before. We define our ORE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{C})$ as follows:

- $\mathcal{K}(1^\lambda, M)$: On input the security parameter and message space $[M]$, the algorithm chooses a key k uniformly at random for F, runs $\Gamma.\mathcal{K}_h$ to obtain the hash and test keys $(\mathsf{hk}, \mathsf{tk})$, and samples a random permutation $\pi : [n] \to [n]$. It sets $\mathsf{ck} \leftarrow \mathsf{tk}$, $\mathsf{sk} \leftarrow (k, \mathsf{hk}, \pi)$ and outputs $(\mathsf{ck}, \mathsf{sk})$.
- $\mathcal{E}(\mathsf{sk}, m)$: On input the secret key SK and a message m, the algorithm computes the binary representation of $m = (b_1, \ldots, b_n)$, and then calculates:

$$u_i = F(k, (i, b_1 b_2 \cdots b_{i-1} || 0^{n-i+1})) + b_i, \quad t_i = \Gamma.\mathcal{H}(\mathsf{hk}, u_i).$$

  Then it sets $v_i = t_{\pi(i)}$ and outputs $\mathsf{CT} = (v_1, \ldots, v_n)$.
- $\mathcal{C}(\mathsf{ck}, \mathsf{CT}_1, \mathsf{CT}_2)$: on input the public parameter, two ciphertexts $\mathsf{CT}_1, \mathsf{CT}_2$ where $\mathsf{CT}_1 = (v_1, \ldots, v_n), \mathsf{CT}_2 = (v'_1, \ldots, v'_n)$, the algorithm runs $\Gamma.\mathcal{T}(\mathsf{tk}, v_i, v'_i)$ for every $i \in [n]$. If there exists i^* such that $\Gamma.\mathcal{T}(\mathsf{tk}, v_{i^*}, v'_{i^*}) = 1$, then the algorithm outputs 1, meaning $m_1 > m_2$; else if there exists a pair i^* such that $\Gamma.\mathcal{T}(\mathsf{tk}, v'_{i^*}, v_{i^*}) = 1$, then the algorithm outputs 0, meaning $m_1 < m_2$; otherwise it outputs it outputs $\perp$, meaning $m_1 = m_2$.

Now, we give the description of the leakage profile, which takes $\boldsymbol{m} = \{m_1, \ldots, m_q\}$ as input and produces:

$$\mathcal{L}'_f(m_1, \ldots, m_q) := (\forall 1 \le i, j, k, l \le q, \mathbf{1}(m_i < m_j), \mathbf{1}(\mathsf{msdb}(m_i, m_j) = \mathsf{msdb}(m_k, m_l)))$$

Compared to $\mathcal{L}_f$, $\mathcal{L}'_f$ gives extra information that $\mathbf{1}(\mathsf{msdb}(m_i, m_j) = \mathsf{msdb}(m_k, m_l))$ even when $i \neq k$. However, $\mathcal{L}'_f$ is still strictly stronger than CLWW, and for any $\boldsymbol{m}$, it's obvious that $\mathcal{L}'_f(\boldsymbol{m}) = \mathcal{L}'_f(2\boldsymbol{m})$, which gives evidence that $\mathcal{L}'_f$ is also smoothed CLWW. And for its simulation based security, applying exactly the same argument as the proof of Theorem 10, we can establish the following theorem.

Theorem 12. *The ORE scheme Π is $\mathcal{L}'_f$-non-adaptive-simulation secure, assuming F is a secure PRF and Γ is restricted-chosen-input secure.*

Therefore, to achieve the privacy of parameter hiding, we can use this efficient scheme as an alternative, such that we only need $O(n)$ pairings for each comparison.

6 PPH from Bilinear Maps

We construct a PPH scheme for the predicate P required in our ORE construction. That is, $P(x, y) = 1$ if and only if $x = y + 1$.

We let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_p$ be a PRF, where p is a prime to be determined at key generation.

Construction. We now define our PPH $\Gamma = (\mathcal{K}_h, \mathcal{H}, \mathcal{T})$.

- $\mathcal{K}_h(1^\lambda)$ This algorithm takes the security parameter as input. It samples descriptions of prime-order p groups $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$, generators $g \in \mathbb{G}, \hat{g} \in \hat{\mathbb{G}}$, a bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. It then chooses $k \xleftarrow{\$} \{0,1\}^\lambda$. It sets the hash key $\mathsf{hk} \leftarrow (k, g, \hat{g})$, the test key $\mathsf{tk} \leftarrow (\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$, a description of the bilinear map and groups, and outputs $(\mathsf{hk}, \mathsf{tk})$.
- $\mathcal{H}(\mathsf{hk}, x)$ This algorithm takes as input the hash key hk, an input x, picks two random non-zero $r_1, r_2 \in \mathbb{Z}_p$ and outputs

$$\mathcal{H}(\mathsf{hk}, x) = (g^{r_1}, g^{r_1 \cdot F(k,x)}, \hat{g}^{r_2}, \hat{g}^{r_2 \cdot F(k,x+1)}).$$

- $\mathcal{T}(\mathsf{tk}, h_1, h_2)$ To test two hash values (A_1, A_2, B_1, B_2) and (C_1, C_2, D_1, D_2), $\mathcal{T}$ outputs 1 if

$$e(A_1, D_2) = e(A_2, D_1),$$

and otherwise it outputs 0.

Hence the domain D is $\{0,1\}^\lambda$ and the range R is $(\mathbb{G}^2, \hat{\mathbb{G}}^2)$

Correctness. Correctness reduces to testing if $F(k, y + 1) = F(k, x)$. If $x = y + 1$ then this always holds. If not, then it is easily shown that finding x, y with this property (and without knowing the key) with non-negligible probability leads to an adversary that contradicts the assumption that F is a PRF.

Security. We prove that PPH is restricted-chosen-input secure, assuming that F is a PRF and that the following assumption holds.

Definition 13. *Let $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ be prime-order p groups, g be generator of $\mathbb{G}$ and $\hat{g}$ be a generator of $\hat{\mathbb{G}}$, tand $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ be a bilinear pairing. We say the* symmetric external Diffie-Hellman assumption holds *with respect to these groups and pairing if for all efficient $\mathcal{A}$,*

$$|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] \Pr[\mathcal{A}(g, g^a, g^b, T) = 1]|$$

and

$$|\Pr[\mathcal{A}(\hat{g}, \hat{g}^a, \hat{g}^b, \hat{g}^{ab}) = 1] \Pr[\mathcal{A}(\hat{g}, \hat{g}^a, \hat{g}^b, T) = 1]|$$

are negligible functions of λ, where a, b, c are uniform over $\mathbb{Z}_p$ and T is uniform over G_T.

We can now state and prove our security theorem.

Theorem 14. *Our PPH Γ is restricted-chosen-input secure, assuming F is a PRF and the SXDH assumption hold with respect to the appropriate groups and pairing.*

Proof. We use a hybrid argument. Let $(A_1, A_2, B_1, B_2) \in \mathbb{G}^2 \times \hat{\mathbb{G}}^2$ denote the challenge hash value given to the adversary during the real game $\mathsf{H}_0 = \mathrm{IND}^{\mathsf{pph}}_{\Gamma,P}(\mathcal{A})$. Additionally, let R be a random element of $\mathbb{G}$, $\hat{R}$ be a random element of $\hat{\mathbb{G}}$, both independent of the rest of the random variables under consideration. Then we define the following hybrid experiments:

- H_1: At the start of the game, a uniformly random function $F^* \xleftarrow{R} \mathsf{Funs}[\{0,1\}^\lambda, \{0,1\}^\lambda]$ is sampled instead of the PRF key K, the rest remain unchanged.
- H_2: The challenge hash value is (A_1, R, B_1, B_2), where $R \xleftarrow{\$} \mathbb{G}$.
- H_3: The challenge hash value is $(A_1, R, B_1, \hat{R})$, where $R \xleftarrow{\$} \hat{\mathbb{G}}$.

In H_3, the adversary is given a random element from the range $\mathcal{R}$. Therefore,

$$\mathsf{Adv}^{\mathsf{pph}}_{\Gamma,P,\mathcal{A}}(\lambda) = |Pr[\mathsf{H}_0 = 1] - Pr[\mathsf{H}_3 = 1]|$$

To prove H_0 is indistinguishable from H_3, we show that each step of the hybrid is indistinguishable from the next. First, it is apparent that H_0 and H_1 are computational indistinguishable by the PRF security, then:

Lemma 15. $\mathsf{H}_1 \approx \mathsf{H}_2$ *under the SXDH assumption.*

Let $\mathcal{A}$ be an adversary playing the PPH security game, and let

$$\epsilon = |Pr[\mathsf{H}_1 = 1] - Pr[\mathsf{H}_2 = 1]|.$$

Then we can build adversary $\mathcal{B}$ that solves SXDH with advantage ϵ. $\mathcal{B}$ is given as input $(g, \hat{g}, B, C)$ and the challenge term T. $\mathcal{B}$ works as follows:

- $\mathcal{B}$ sets $\mathsf{tk} = (\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ and sends it to $\mathcal{A}$. After receiving $x^* \xleftarrow{\$} \mathcal{A}(\mathsf{tk})$ it simulates a random function F^* via lazy sampling, and it will implicitly set $F^*(x^*) = b$, the discrete logarithm of B. It prepares the challenge as by selecting $r^* \xleftarrow{\$} \mathbb{Z}_p$ and computing

  $$A_1 = g^c, A_2 = T, B_1 = \hat{g}^{r^*}, B_2 = \hat{g}^{r^* F^*(x^*+1)}$$

  and runs $\mathcal{A}$ on input $\mathsf{tk}, x^*, (A_1, A_2, B_1, B_2)$.
- To answer hash query for $x \neq x^*$ from $\mathcal{A}$, $\mathcal{B}$ calculates $F^*(x)$ and $F^*(x+1)$ (note that $x, x+1 \neq x^*$). Then $\mathcal{B}$ picks r_1, r_2 randomly and computes:

  $$\mathcal{H}(x) = g^{r_1}, g^{r_1 \cdot F^*(x)}, \hat{g}^{r_2}, \hat{g}^{r_2 \cdot F^*(x+1)};$$

  If $\mathcal{A}$ queries $x = x^*$, $\mathcal{B}$ calculates $F^*(x^*+1)$, picks $r'_1, r'_2 \xleftarrow{\$} \mathbb{Z}_p$, and computes

  $$\mathcal{H}(x^*) = g^{r'_1}, B^{r'_1}, \hat{g}^{r'_2}, \hat{g}^{r'_2 \cdot F^*(x^*+1)};$$

– Finally $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

We note that in $\mathcal{A}$'s view, without querying $\mathcal{A}(x^* - 1)$, $\mathcal{B}$ simulates the game properly. If $T = g^{bc}$, then $\mathcal{B}$ simulates H_1, and if T s random then it simulates H_2. Hence if $\mathcal{A}$ has an advantage ϵ in distinguishing H_1 and H_2, then $\mathcal{B}$ has the same advantage to break SXDH assumption.

We also have the following lemma:

Lemma 16. $\mathsf{H}_2 \approx \mathsf{H}_3$ *under the SXDH assumption.*

The proof is exactly the same as the prior hybrid step, except in the group $\hat{\mathbb{G}}$ part of the hash instead of $\mathbb{G}$. We omit the details.

Collecting the steps completes the proof of Theorem 14.

Acknowledgments. David Cash is supported by NSF CNS-1453132. Feng-Hao Liu is supported by NSF CNS-1657040. Adam O'Neill is supported in part by NSF CNS-1650419. Mark Zhandry is supported by NSF. David Cash and Cong Zhang are partially supported by DARPA and SSC Pacific under contract N66001-15-C-4070. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DARPA or SSC Pacific.

References

1. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: SIGMOD (2004)
2. Arasu, A., Blanas, S., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R., Venkatesan, R.: Orthogonal security with cipherbase. In: CIDR (2013)
3. Arasu, A., Eguro, K., Kaushik, R., Ramamurthy, R.: Querying encrypted data (tutorial). In: ICDE (2013)
4. Bajaj, S., Sion, R.: TrustedDB: a trusted hardware-based database with privacy and data confidentiality. TKDE **26**(3), 752–765 (2014)
5. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_30
6. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_13
7. Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_33
8. Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., Zimmerman, J.: Semantically secure order-revealing encryption: multi-input functional encryption without obfuscation. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 563–594. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_19
9. Canetti, R.: Towards realizing random oracles: hash functions that hide all partial information. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 455–469. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052255

10. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: CCS (2015)
11. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_20
12. Cash, D., Liu, F.-H., O'Neill, A., Zhandry, M., Zhang, C.: Parameter-hiding order revealing encryption. Cryptology ePrint Archive, Report 2018/698 (2018). https://eprint.iacr.org/2018/698
13. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_30
14. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 577–594. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_33
15. Chenette, N., Lewi, K., Weis, S.A., Wu, D.J.: Practical order-revealing encryption with limited leakage. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 474–493. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_24
16. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 3–12. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_1
17. Coron, J.-S., Lee, M.S., Lepoint, T., Tibouchi, M.: Cryptanalysis of GGH15 multilinear maps. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 607–628. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_21
18. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: CCS (2006)
19. Dautrich Jr., J.L., Ravishankar, C.V.: Compromising privacy in precise query protocols. In: EDBT (2013)
20. Durak, F.B., DuBuisson, T.M., Cash, D.: What else is revealed by order-revealing encryption? In: ACM CCS (2016)
21. Goh, E.-J., et al.: Secure indexes. IACR Cryptology ePrint Archive 2003:216 (2003)
22. Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M., Ristenpart, T.: Leakage-abuse attacks against order-revealing encryption. Cryptology ePrint Archive, Report 2016/895 (2016). http://eprint.iacr.org/2016/895
23. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, SIGMOD 2002, pp. 216–227. ACM, New York (2002)
24. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. VLDBJ **21**(3), 333–358 (2012)
25. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: NDSS (2012)
26. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Inference attack against encrypted range queries on outsourced databases. In: CODASPY (2014)
27. Joye, M., Passelègue, A.: Function-revealing encryption (2016)
28. Kamara, S., Moataz, T.: SQL on structurally-encrypted databases. Cryptology ePrint Archive, Report 2016/453 (2016). http://eprint.iacr.org/

29. Lewi, K., Wu, D.J.: Order-revealing encryption: new constructions, applications, and lower bounds. In: ACM CCS (2016)
30. Liu, C., Zhu, L., Wang, M., Tan, Y.-A.: Search pattern leakage in searchable encryption: attacks and new construction. Inf. Sci. **265**, 176–188 (2014)
31. Mandal, A., Roy, A.: Relational hash: probabilistic hash for verifying relations, secure against forgery and more. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 518–537. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_25
32. Miles, E., Sahai, A., Zhandry, M.: Annihilation attacks for multilinear maps: cryptanalysis of indistinguishability obfuscation over GGH13. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 629–658. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_22
33. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_30
34. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: CCS (2015)
35. Pandey, O., Rouselakis, Y.: Property preserving symmetric encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 375–391. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_23
36. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: SOSP (2011)
37. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: SP (2000)