On Locally Decodable Codes in Resource Bounded Channels

Jeremiah Blocki
Purdue University, West Lafayette, IN, USA
jblocki@purdue.edu

Shubhang Kulkarni
Purdue University, West Lafayette, IN, USA
kulkar17@purdue.edu

Samson Zhou
Carnegie Mellon University, Pittsburgh, PA, USA
samsonzhou@gmail.com

Abstract

Constructions of locally decodable codes (LDCs) have one of two undesirable properties: low rate or high locality (polynomial in the length of the message). In settings where the encoder/decoder have already exchanged cryptographic keys and the channel is a probabilistic polynomial time (PPT) algorithm, it is possible to circumvent these barriers and design LDCs with constant rate and small locality. However, the assumption that the encoder/decoder have exchanged cryptographic keys is often prohibitive. We thus consider the problem of designing explicit and efficient LDCs in settings where the channel is *slightly* more constrained than the encoder/decoder with respect to some resource e.g., space or (sequential) time. Given an explicit function f that the channel cannot compute, we show how the encoder can transmit a random secret key to the local decoder using $f(\cdot)$ and a random oracle $\mathsf{H}(\cdot)$. We then bootstrap the private key LDC construction of Ostrovsky, Pandey and Sahai (ICALP, 2007), thereby answering an open question posed by Guruswami and Smith (FOCS 2010) of whether such bootstrapping techniques are applicable to LDCs in channel models weaker than just PPT algorithms. Specifically, in the random oracle model we show how to construct explicit constant rate LDCs with locality of polylog in the security parameter against various resource constrained channels.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Locally Decodable Codes, Resource Bounded Channels

Digital Object Identifier 10.4230/LIPIcs.ITC.2020.16

Related Version `https://arxiv.org/pdf/1909.11245.pdf`

Funding This research was supported in part by the National Science Foundation (CCF Award #1910659).

Acknowledgements We would like to thank anonymous reviewers for helpful feedback that improved the presentation of this paper.

1 Introduction

Consider the classical one-way communication setting where two parties, the *sender* and *receiver*, communicate over a *noisy channel* that may *corrupt* parts of any message sent over it. An *error correcting code* is an invertible transformation mapping messages into *codewords* that are then transmitted over the noisy channel. The goal is to ensure that the decoder can (w.h.p.) reliably recover the entire message from the corrupted codeword. For locally decodable codes (LDCs) we have an even stronger goal: The decoder should be able to reliably recover *any* individual bit of the original message (w.h.p.) by examining at most ℓ bits of the corrupted codeword. An ideal LDC should have a good rate (i.e., the codeword should not be much longer than the original message) and small locality ℓ.

Historically, there have been two major lines of work associated with modelling the channel behavior. In Shannon's *symmetric channel* model, the channel each bit of the codeword independently at random with some fixed probability. By contrast, in Hamming's *adversarial channel* model the channel corrupts the codeword in a worst case manner subject to an upper bound on the total number of corruptions.

Unsurprisingly, when we work in Shannon's channel model it is much easier to design LDCs with good rate/locality. By contrast, state of the art LDC constructions for Hamming channels either have very high locality e.g., $\ell = 2^{\mathcal{O}\left(\sqrt{\log n \log\log n}\right)}$ [29] or poor rate e.g., Hadamard codes have constant locality $\ell = \mathcal{O}(1)$ but the codeword has exponential length. Unfortunately, in many real-world settings independent random noise is not a realistic model of channel behavior e.g., burst-errors are common in reality, but unlikely in Shannon's model. Thus, coding schemes designed to work in Shannon's channel model are not necessarily suitable in practice. By contrast, coding schemes designed to work in Hamming's adversarial setting must be able to handle *any* error pattern.

Our central motivating goal is to find classes of adversarial channels that are expressive enough to model any error patterns that would arise in nature, yet admit LDCs with good decoding algorithms. LDCs have found remarkable applications throughout various fields, notably private information retrieval schemes [9, 16, 30], psuedo-random generator constructions [7, 38], self-correcting computations [18, 21], PCP systems [8] and fault tolerant storage systems [27].

Lipton [31] introduced the *adversarial computationally bounded* model, where the channel was viewed as a Hamming channel restricted to bounded corruption by a *probabilistic polynomial time* (PPT) algorithm. The notion of adversaries being computationally bounded is well-motivated by real-world channels that have some sort of limitations on their computations i.e., we expect error patterns encountered in nature to be modeled by some (possibly unknown) PPT algorithm. We argue that even Lipton's channel significantly overestimates the capability of the channel. For example, if the channel has reasonably small latency, say 10 seconds, and the world's fastest single core processor can evaluate 10 billion instructions per second then the depth of any (parallel) computation performed by the channel is at most 100 billion operations.

This view of modelling the channel as more restricted than just PPT was further explored by Guruswami and Smith [23] who studied channels that could be described by simple (low-depth) circuits. Remarkably, even such a simple restriction allowed them to design codes that enjoyed no public/private key setup assumptions, while matching the Shannon capacity using polynomial time encoding/decoding algorithms. With such positive results, it is natural to ask whether similar results may be expected for LDCs.

1.1 Contributions

We introduce *resource bounded adversarial channel* models which admit LDCs with good locality whilst still being expressive enough to plausibly capture any error pattern for most real-world channels. We argue that these resource bounded channel models are already sufficiently expressive to model any corruption pattern that might occur in nature e.g., burst-errors, correlated errors. For example, observe that the channel must compute the entire error pattern *before* the codeword is delivered to the receiver. Thus, the channel can be viewed as *sequentially time bounded* e.g., the channel may perform arbitrary computation in parallel but the total depth of computation is bounded by the latency of the channel. The notion of a space bound (or space-time bound) channel can be similarly motivated.

We introduce *safe functions* as a general way to characterize LDC friendly channels. Intuitively, a function f is "safe" for a class of channels if the channel is not able to predict $f(x)$ given x. We show how to construct safe functions for several classes of resource bounded channels including time bounded, space bounded, and cumulative memory cost bounded channels in the parallel random oracle model. For example, in the random oracle model the function $\mathsf{H}^{t+1}(x)$ is a safe function for the class of sequentially time-bounded adversaries i.e., it is not possible to evaluate the function using fewer than t sequential calls to the random oracle H. We also discuss how to construct safe functions for the class of space (resp. space-time) bounded channels using random oracles.

Furthermore, we give a general framework for designing good locally decodable codes against resource bounded adversarial channels by using safe functions to bootstrap existing private-key LDC constructions. Our framework assumes no a priori private or public key setup assumptions, and constructs explicit LDCs over the binary alphabet[1] with constant rate against *any* class of resource bounded adversaries admitting *safe functions*.

Our local decoder can decode correctly with arbitrarily high constant probability after examining at most $\mathcal{O}\left(f(\kappa)\right)$ bits of the corrupted codeword, where κ is the security parameter[2] and $f(\kappa)$ is any function such that $f(\kappa) = \omega(\log \kappa)$ e.g., $f(\kappa) = \log^{1+\varepsilon} \kappa$ or $f(\kappa) = \log \kappa \log \log \kappa$. By contrast, state of the art LDC constructions for Hamming channels have very high locality e.g., $2^{\mathcal{O}\left(\sqrt{\log n \log \log n}\right)}$ [29]. Our codes are robust against a constant fraction of corruptions, and are (essentially) *non-adaptive* i.e., the local decoding algorithm can decode after submitting just two batches of queries.

Our constructions stand at the intersection of coding theory and cryptography, using well-known tools and techniques from cryptography to provide notions of (information theoretic) randomness and security for communication protocols between sender/receiver. To prove the security of our constructions, we introduce a *two-phase distinguisher hybrid argument*, which may be of independent interest for other coding theoretic problems in these resource bounded channel models.

1.2 Technical Overview

Private LDCs. Our starting point is the private locally decodable codes of [33]. These LDCs permit nearly optimal query complexity, asymptotically positive rate and reliable decoding with high probability, but make the strong assumption that the sender and receiver have already exchanged a secret key sk that is unknown to the PPT adversarial channel over which they communicate. In our setting the sender and the receiver do not have access to any secret key. Our constructions thus *reduce* the general setting (no setup assumptions) against resource bounded channels to the shared private key setting against these channels, so that we can bootstrap private LDC constructions.

---

1 Note that small alphabet sizes are attractive for practical channels designed to transmit bits efficiently.

2 In this paper we use the security parameter κ in an asymptotic sense e.g., for any attacker running in time $\mathtt{poly}(\kappa)$ there is a negligible function $\mathtt{negl}(\kappa)$ upper bounding the probability that the attacker succeeds. In particular, the function $\mathtt{negl}(\kappa) = 2^{-\log^{1+\varepsilon} \kappa)}$ is negligible, but does not provide κ-bits of concrete security i.e., any attacker running in time t succeeds with probability at most $t2^{-\kappa}$.

Bootstrapped Encoder/Decoder. Our encoder uses the following high level template: (1) samples a random seed r (2) computes a predetermined *safe function* $f(r)$ on the seed and extracts a secret key sk from $f(r)$ (e.g., using a random oracle) (3) Uses the private LDC encoder to encode the message using sk (4) appends a reliable encoding (error-correcting code composed with a repetition code) of the random seed r to the codeword. The local decoder (1) decodes the random seed r (random sampling + majority vote). (2) Evaluates the safe function $f(r)$ to recover the secret key sk. (3) Uses the private LDC decoder with the secret key sk to recover the desired bit of the original message.

Security Proof. We remark that there are a few subtle challenges that arise while proving the security of our bootstrapped constructions. We want to prove that the channel fails to produce a corrupted codeword that fools the local decoding algorithm. Towards this goal we might try to prove that the channel cannot distinguish the derived key sk from a truly random key, even when given the nonce r. However, this is insufficient to prove that the local decoder is successful because the local decoder *is* able to recover sk from the f. We introduce a novel *two-phase distinguisher game* to address these challenges. In particular, we consider an attacker-distinguisher pair who tries to predict whether or not the secret encoding key sk is derived from the nonce r ($b = 0$) or was selected uniformly at random ($b = 1$). In phase 1 the (resource bounded) attacker generates a corrupted codeword which is given to the distinguisher in phase 2 who must then guess whether $b = 1$ or $b = 0$. The distinguisher is computationally unbounded, but is not allowed to query the random oracle. If f is a safe function then the advantage of any such attacker-distinguisher pair can be shown to be negligible. We demonstrate that any channel which succeeds at fooling our local decoder yields an attacker-distinguisher pair for this two phase game – the distinguisher works by simulating the private LDC decoder to distinguish between the two aforementioned encodings. It follows that the channel cannot fool the local decoder (except with negligible probability).

1.3 Related Work

Many existing code constructions consider an underlying channel that can only introduce a bounded number of errors, but has an unlimited time to adversarially decide the positions of these errors. These codes are therefore resilient to any possible error pattern with a bounded number of corruptions, corresponding to Hamming's error model, and are safe for data transmission. However, this resiliency to the worst-case error leads to coding limitations and some possibly undesirable tradeoffs. On one hand, current constructions for LDCs that focus on efficient encoding can obtain any constant rate $R < 1$ while simultaneously being robust to any constant fraction $\delta < 1 - R$ of errors and using $2^{\mathcal{O}\left(\sqrt{\log n \log\log n}\right)}$ queries for decoding [29]. On the other hand, codes that focus on low query complexity obtain blocklength that is subexponential in the message length while using a constant number of queries $q \leq 3$ [39, 20, 19]. Finally, if exactly $q = 2$ queries are desired, any code *must* use blocklength exponential in the message length [28]. Avoiding such drastic tradeoffs between blocklength and query complexity would be attractive for other natural channels in contrast to Hamming's error model. For example, Shannon introduces a model in which each symbol has some independent probability of being corrupted; this probability is generally fixed across all symbols and known a priori. However, this probabilistic channel may be too weak to capture natural phenomenon such as bursts of consecutive error.

Thus it is reasonable to believe that many natural channels lie between these two extremes; in particular, Lipton [31] argues that many reasonable channels are computationally bounded and can be modeled as PPT algorithms. In this model, [31] introduced an analog to classical error-correcting codes that is robust to a fraction of errors beyond the rates provably tolerable by *any* code in the adversarial Hamming channel model. Similarly, a line of work [31, 32, 23, 37] have improved upon the error rate limits of classical error-correcting codes in slight variants of Lipton's computationally bounded channel model. A weakness of the codes introduced by [31] is the strong cryptographic assumption that the sender and receiver share a *secret* random string unknown to the channel. This weakness is ameliorated by [32], who observe that if a message is encoded by digitally signing a code that is *list-decodable* with a secret key, then an adversarial PPT is unlikely to produce valid signatures. Conversely, the decoder can select the unique message from the list of possible messages with a valid signature, effectively producing public-key error-correcting codes against computationally bounded channels. Subsequently, [23] further removes the public-key setup assumption specifically for the channel in which either the error is independent of the actual message being sent, or the errors can be described by polynomial size circuits. Their results are based on the idea that the sender can choose a permutation and some key that is computable by the decoder but not by the channel, since it operates with low complexity. In some loose sense, their results are an example of our framework when the channel has bounded circuit complexity, i.e. the bounded resource is circuit complexity of the error.

[33] obtain LDCs with constant information and error rates over the binary alphabet against computationally bounded errors, using a small number of queries to the corrupted word; specifically they can achieve any $\omega(\log \kappa)$ query complexity, where κ is the desired security parameter. However, their results not only assume the existence of one-way functions, but also once again assume a predetermined private key known to both the encoder and decoder but not the channel, similar to [31]. Analogous to the improvements of [32] for classical error codes, [24, 25] construct public-key LDCs, assuming the existence of Φ-hiding schemes [15] and IND-CPA secure cryptosystems.

Ben-Sasson et al. [10] introduce the concept of *relaxed locally decodable codes* (RLDCs) as an alternative means of decreasing the tradeoffs between rate and locality in classical LDCs. In contrast to LDCs, the decoding algorithm for RLDCs is allowed to output $\perp$ sometimes to reveal that the correct value is unknown, though it is limited in the fraction of outputs in which it can output $\perp$. The RLDCs proposed by Ben-Sasson et al. [10] obtain constant query complexity and blocklength $n = k^{1+\epsilon}$. Subsequently, Gur et al. [22] construct *relaxed locally correctable codes* (RLCCs) with attractive properties but significant tradeoffs; they propose codes with constant query complexity and error rate but block length roughly quartic in the message length as well as codes with constant error rate and linear block length, but quasipolynomial ($(\log n)^{\mathcal{O}(\log \log n)}$) query complexity. These parameters are significantly better than classical locally correctable codes and their results immediately extend to RLDCs, since the original message is embedded within the initial part of the encoding. However, these tradeoffs are still undesirable.

Recently, Blocki et al. [12] study RLDCs and RLCCs on adversarial but computationally bounded channels in an effort to reduce these tradeoffs. They obtain RLDCs and RLCCs over the binary alphabet, with constant information rate, and poly-logarithmic locality. Moreover, their codes require no public-key or private-key cryptographic setup; the only setup assumption required is the selection of the public parameters (seed) for a collision-resistant hash function.

2 Preliminaries

2.1 Notation

We use the notation $[n]$ to represent the set $\{1, 2, \ldots, n\}$. For any $x, y \in \Sigma^n$, let $\mathsf{HAM}(x)$ denote the Hamming weight of x, i.e. the number of non-zero coordinates of x. Let $\mathsf{HAM}(x, y) = \mathsf{HAM}(x - y)$ denote the Hamming distance between the vectors x and y. All logarithms will be base 2. For n vectors $x_1, \ldots, x_n$, we use $\mathsf{majority}(x_1 \cdots x_n)$ to denote the vector that appears most frequently. If such a vector is not unique, then an arbitrary vector of highest frequency is chosen. For any vector $x \in \Sigma^n$, let $x[i]$ be the i^{th} coordinate of x. We also let $x \circ y$ denote the concatenation of x with y and $x \oplus y$ denote the bitwise XOR of x and y. For a randomized function $f(\cdot)$, the notation $f(\cdot; R)$ will be used to denote that $f(\cdot)$ uses random coins R as its randomness. A function $\mathsf{negl}(\kappa)$ is said to be *negligible* in κ if $\mathsf{negl}(\kappa) \in o\left(\left|\frac{1}{\mathsf{poly}(\kappa)}\right|\right)$ for any non-zero polynomial $\mathsf{poly}(\cdot)$. Finally, we distinguish between inputs and parameters to a function f as follows: $f(\text{inputs} \cdots)[\text{parameters} \cdots]$.

2.2 Locally Decodable Codes

We consider the setting where sender $\mathcal{S}$ encodes a *message* x into a *codeword* y using an *encoding algorithm* so that y is sent over noisy channel $\mathcal{C}$, which then hands over the possibly corrupted codeword y' to $\mathcal{R}$, who then uses a *decoding algorithm* to obtain the original message. We denote $x \in \Sigma^k$ and $y \in \Sigma^K$ where Σ is the alphabet. We denote the alphabet size by $q = |\Sigma|$. We consider the model where y' corresponds to y with some symbols replaced with others in Σ. The term *corruptions* refers to such symbol replacements within y, with a single corruption meaning a single symbol replacement, so that $y' \in \Sigma^K$. The encoding and decoding algorithms are denoted by $\mathsf{Enc} : \Sigma^k \to \Sigma^K$ and $\mathsf{Dec} : \Sigma^K \to \Sigma^k$. We use the terms sender, encoder, and encoding algorithm interchangeably, and similarly for receiver, decoder, and decoding algorithm.

A *code* is an encoder-decoder pair. The *information rate* or simply *rate* of the code is the ratio k/K, so that a lower rate corresponds to a larger amount of information redundancy introduced by the code. The message length, codeword length, and alphabet size characterize a *coding scheme*. Coding schemes with high rate and low alphabet size are desired.

An error correcting code allows the decoder to recover the entire original message x by reading the entire y'. It is also possible to construct codes that only need to read a few symbols of y' rather than the entire message to recover a small part of the message. Such codes are called *locally decodable codes* (LDC), and will be the focus of this work. An LDC has *locality* ℓ, *error rate* ρ and *error correction probability* p if any character of x may be recovered with probability at least p by making at most ℓ queries to y', even when the channel corrupts ρ fraction of all symbols of y to generate y'. We use the terms *query complexity* and locality interchangeably. When ρ and p are clear from context (as constants), the scheme may be referred to as an ℓ-LDC. Naturally, LDCs with low locality, high error rate, and high error correction probability are desired.

2.3 Definitions

The focus of this work will be the construction of LDCs (Section 2.4) for *resource-bounded* channels (Section 4.1). In this section, we present several building blocks that we will require in our constructions – LDC*s, *private*-LDCs and *safe functions*. We first give two classical definitions pertaining to LDCs that compactly summarize our discussion in Section 2.2.

▶ **Definition 1.** *A* $(K,k)_q$-coding scheme $C[K,k,q] = (\mathsf{Enc},\mathsf{Dec})$ *is a pair of encoding* $\mathsf{Enc}:\Sigma^k \to \Sigma^K$ *and decoding* $\mathsf{Dec}:\Sigma^K \to \Sigma^k$ *algorithms where* $|\Sigma| = q$*. The* information rate *of the scheme is defined as* $\frac{k}{K}$*.*

▶ **Definition 2.** *A* $(K,k)_q$*-coding scheme* $C[K,k,q] = (\mathsf{Enc},\mathsf{Dec})$ *is an* (ℓ,ρ,p)-locally decodable code (LDC) *if* Dec*, with query access to a word* y' *such that* $\mathsf{HAM}(\mathsf{Enc}(x),y') \le \rho K$*, on input index* $i \in [k]$*, makes at most* ℓ *queries to* y' *and outputs* x_i *with probability at least* p *over the randomness of the decoder.*

Next, we present a simple variant of LDCs which we denote by LDC*s. These will be very similar to LDCs except that they are required to decode the entire original message while making as few queries to the corrupted codeword as possible. They are defined with respect to the same setting as in Section 2.2.

▶ **Definition 3.** *A* $(K,k)_q$*-coding scheme* $C[K,k,q] = (\mathsf{Enc},\mathsf{Dec})$ *is an* (ℓ,ρ,p)-LDC* *if* Dec*, with query access to a word* y' *such that* $\mathsf{HAM}(\mathsf{Enc}(x),y') \le \rho K$*, makes at most* ℓ *queries to* y' *and outputs* x *with probability at least* p *over the randomness of the decoder.*

We remark that it will be typically desired that for an LDC* $\mathsf{C}[K,k,q]$, the locality be $\mathcal{O}(k)$ even when K is very large. We now move on to define *one-time private*-LDC*s* analogous to Definition 2 as an alternative to that given by Ostrovsky, Pandey and Sahai [33]. Ostrovsky et al. also give explicit constructions of one-time private-LDCs, and so we restate their result according to our new definition. Refer to the full version [13] for an overview of [33].

$\mathtt{priv-LDC-Sec-Game}[\mathcal{A},x,\kappa,\rho,p]$:

1. The challenger generates a secret key $\mathsf{sk} \leftarrow \mathsf{GenKey}(1^\kappa)$, computes the codeword $y \leftarrow \mathsf{Enc}(x,\kappa,\mathsf{sk})$ for the message x and sends the codeword y to the attacker.
2. The attacker outputs a corrupted codeword $y' \leftarrow \mathcal{A}(x,y,\kappa,\rho,p,k,K)$ where $y' \in \Sigma^K$ should have hamming distance at most ρK from y.
3. The output of the experiment is determined as follows:

$$\mathtt{priv-LDC-Sec-Game}[\mathcal{A},x,\kappa,\rho,p] = \begin{cases} 1 & \text{if } \exists i \le k \text{ s.t. } \Pr[\mathsf{Dec}^{y'}(i,\kappa,\mathsf{sk}) = x_i] < p \\ 0 & \text{otherwise} \end{cases}$$

If the output of the experiment is 1 (resp. 0), the attacker $\mathcal{A}$ is said to *win* (resp. *lose*) against C.

Figure 1 $\mathtt{priv-LDC-Sec-Game}$ defining the interaction between an attacker and an honest party.

▶ **Definition 4** (One-Time Private-LDC)**.** *A triplet of probabilistic algorithms* $\mathsf{C}[K,k,\kappa] = (\mathsf{GenKey},\mathsf{Enc},\mathsf{Dec})$ *is an* $(\ell,\rho,p,\epsilon,\mathbb{C})$-private locally decodable code (private LDC) *against a class* $\mathbb{C}$ *if* Dec *makes at most* ℓ *queries and for all attackers* $\mathcal{A} \in \mathbb{C}$ *and all messages* $x \in \Sigma^k$*,*

$$\Pr[\mathtt{priv-LDC-Sec-Game}[\mathcal{A},x,\kappa,\rho,p] = 1] \le \epsilon$$

where the probability is taken over the random coins of $\mathcal{A}$ *and* GenKey*. If* $\mathbb{C}$ *is the set of all (computationally unbounded) attackers we say that the scheme is a* (ℓ,ρ,p,ϵ)-private LDC*.*

A construction of Ostrovsky et al. [33] yields a constant-rate One-Time Private-LDC with constant rate, low locality $\ell = \omega(\log\kappa)$ and negligible failure probability whenever $k = \mathrm{poly}(\kappa)$.

▶ **Theorem 5** (One-Time Private-LDC Existence). *[33] Let $f(\kappa)$ be any function such that $f(\kappa) = \omega(\log \kappa)$. Then, for security parameter κ and for all $K > k > 0$ such that $k = \mathrm{poly}(\kappa)$ where* poly *is any non-zero polynomial, there exists a $(K, k)_2$ coding scheme that is a one-time $(\ell_{\mathsf{OPS}}, \rho_{\mathsf{OPS}}, p_{\mathsf{OPS}}, \epsilon_{\mathsf{OPS}})-$private* LDC *where $\ell_{\mathsf{OPS}} = f(\kappa)$, ρ_{OPS} is a constant, $p_{\mathsf{OPS}} = 1$, and $\epsilon_{\mathsf{OPS}} \leq k\left(\frac{e}{4}\right)^{-\rho_{\mathsf{OPS}}\ell_{\mathsf{OPS}}}$ is negligible in the security parameter.*

Our contributions in the subsequent sections will assume that the coding scheme and channel all have access to a random oracle. Furthermore, we assume that the channel is a pROM algorithm with respect to this random oracle (refer to the initial discussion in Section 4.1 for an overview of the pROM model). The following definition establishes a notion of *privacy* against classes (i.e. sets) of adversarial channels in terms of "hard to compute" functions.

▶ **Definition 6** (Safe Function). *We say that a function $f : \{0,1\}^n \to \{0,1\}^*$ is δ-*safe *for a class $\mathbb{C}$ of algorithms if for all $\mathcal{A} \in \mathbb{C}$ we have*

$$\Pr\left[\mathcal{A}(x) = f(x)\right] \leq \delta$$

where the probability is taken over the random coins of $\mathcal{A}$ and the selection of an input $x \in \{0,1\}^n$. If the function $f = f^{\mathsf{H}(\cdot)}$ is defined using a random oracle, then the probability $\Pr\left[\mathcal{A}^{\mathsf{H}(\cdot)}(x) = f^{\mathsf{H}(\cdot)}(x)\right]$ is also taken over the selection of the random oracle $\mathsf{H}(\cdot)$.

We will use the notation $\mathbb{S}_{\mathbb{C}}$ to denote a $\delta-$safe function for class $\mathbb{C}$. In the above definition, we usually think of δ as being a negligibly small parameter. We remark that in the parallel random oracle model, one can construct functions with sharp thresholds on the required resources. For example, the function $\mathsf{H}^{t+1}(x)$ is trivial to compute using at most $t+1$ sequential queries to $\mathsf{H} : \{0,1\}^* \to \{0,1\}^2$, but *any* parallel algorithm making at most q queries over t rounds succeeds with probability at most $\delta = (t^2 + tq)/2^w$.

Precomputation. Definition 6 can be extended to consider an attacker who is allowed to perform precomputation with the random oracle $\mathsf{H}(\cdot)$ before receiving the input x. In particular, we could consider a pair of oracle algorithms $(\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1^{\mathsf{H}(\cdot)}(m)$ outputs an m-bit hint $\sigma \in \{0,1\}^m$ for $\mathcal{A}_2$ after making at most q queries to $\mathsf{H}(\cdot)$. We could modify the definition to require that for all $\mathcal{A}_2 \in \mathbb{C}$ we have

$$\Pr\left[\mathcal{A}_2^{\mathsf{H}(\cdot)}(x, \mathcal{A}_1^{\mathsf{H}(\cdot)}(m)) = f^{\mathsf{H}(\cdot)}(x)\right] \leq \delta$$

where the randomness is taken over the selection of x, the random oracle $\mathsf{H}(\cdot)$, and the random coins of $\mathcal{A}_2$. Here, $\mathcal{A}_1^{\mathsf{H}(\cdot)}(m)$ (precomputation) is not necessarily constrained to be in the same class $\mathbb{C}$ as $\mathcal{A}_2$.

We remark that for $k = m/w$, a precomputing attacker can succeed with probability at least $k/2^n$ by having $\mathcal{A}_1^{\mathsf{H}(\cdot)}(m)$ output the hint $\sigma = f^{\mathsf{H}(\cdot)}(1), \ldots, f^{\mathsf{H}(\cdot)}(k)$. Then $\mathcal{A}_2^{\mathsf{H}(\cdot)}(x, \sigma)$ first checks if $x \in \{1, \ldots, k\}$ and, if so, simply returns the output $f^{\mathsf{H}(\cdot)}(x)$ which is already recorded in the hint σ. Thus, we need the length n of the random nonce x to be sufficiently large to resist brute-force precomputation attacks. By contrast, if the attacker does not get to perform any precomputation then δ can be negligible even when $n = \mathcal{O}(1)$.

All of the safe functions we consider would also be secure under this stronger notion. For example, $\mathsf{H}^{t+1}(x)$ is δ-safe for $\delta = \mathcal{O}\left((qt + t^2)/2^w + qt/2^n\right)$ where x is a random n bit string, $\mathcal{A}_1$ makes at most q total random oracle queries, and $\mathcal{A}_2$ makes at most q total queries in at most t rounds to $\mathsf{H}(\cdot)$. In our LDC constructions we select a random nonce of length $\Omega(\log^{1+\varepsilon} \kappa)$ to ensure that a precomputing attacker fails.

2.4 Our Model

We first define an experiment to model the interaction between a code and an algorithm from a class of pROM algorithms adversarial against the code. For random oracle $\mathsf{H}(\cdot)$, let $\mathsf{C} = (\mathsf{Enc}^{\mathsf{H}(\cdot)}, \mathsf{Dec}^{\mathsf{H}(\cdot)})$ be a $(K,k)_q$-coding scheme in the random oracle model and let $\mathbb{C}$ be a class of pROM algorithms. Then, the interaction of $\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}$ having error rate ρ, with the code C is defined in Figure 2 (analogous to `priv-LDC-Sec-Game` defined in Figure 1). Here, the security parameter κ, and the decoding probability p are also given as inputs to the game. We now formally define a notion of LDCs analogous to Definition 2, but with respect to general classes of adversarial (pROM) channels.

$\mathtt{LDC-Sec-Game}[\mathcal{A}, x, \mathsf{H}, \kappa, \rho, p]$:

1. The challenger computes $y \leftarrow \mathsf{Enc}^{\mathsf{H}(\cdot)}(x, \kappa)$ encoding the message x and sends $y \in \Sigma^K$ to the attacker.
2. The channel $\mathcal{A}^{\mathsf{H}(\cdot)}$ outputs a corrupted codeword $y' \leftarrow \mathcal{A}^{\mathsf{H}(\cdot)}(x, y, \kappa, \rho, p, k, K)$ where $y' \in \Sigma^K$ should have hamming distance at most ρK from y.
3. The output of the experiment is determined as follows:

$$\mathtt{LDC-Sec-Game}[\mathcal{A}, x, \mathsf{H}, \kappa, \rho, p] = \begin{cases} 1 & \text{if } \exists i \leq k \text{ such that } \Pr[\mathsf{Dec}^{y', \mathsf{H}(\cdot)}(i, \kappa) = x_i] < p \\ 0 & \text{otherwise} \end{cases}$$

If the output of the experiment is 1 (resp. 0), the channel is said to *win* (resp. *lose*).

Figure 2 $\mathtt{LDC-Sec-Game}$ defining the interaction between an attacker and an honest party.

▶ **Definition 7.** *Let $\mathbb{C}$ be a class of pROM algorithms. A $(K,k)_q$-coding scheme $\mathsf{C}[K,k,q] = (\mathsf{Enc}^{\mathsf{H}(\cdot)}, \mathsf{Dec}^{\mathsf{H}(\cdot)})$ is an $(\ell, \rho, p, \epsilon, \mathbb{C})$-locally decodable code (LDC) if $\mathsf{Dec}^{\mathsf{H}(\cdot)}$ makes at most ℓ queries and for all $\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}$ and all messages $x \in \Sigma^k$ we have*

$$\Pr[\mathtt{LDC-Sec-Game}[\mathcal{A}, x, \mathsf{H}, \kappa, \rho, p] = 1] \leq \epsilon$$

where the probability is taken over the random coins of $\mathcal{A}^{\mathsf{H}(\cdot)}$ and the selection of the random oracle H.

3 Constructions

We begin by discussing the use of safe functions in Section 3.1 and give several examples of constructing such functions in Section 4. We then show how allowing an encoder/decoder pair with enough resources to compute safe functions can effectively generate a random shared secret key between the pair. This secret key can then be bootstrapped into existing private LDC constructions to give codes against resource bounded adversaries. We give our final framework in Section 3.2 and the main proofs in Sections 3.3 and 3.4.

3.1 Using Safe Functions

Let $\mathbb{C}$ be a class of algorithms with safe function $\mathbb{S}_{\mathbb{C}}$. For some input $x \in \{0,1\}^n$ to $\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}$, we will be interested in bounding the probability of the undesirable event where the $\mathcal{A}^{\mathsf{H}(\cdot)}$queries the random oracle at any string of the form $y \circ \mathbb{S}_{\mathbb{C}}(x)$ with $y \in \{0,1\}^{\lceil \log_2 \alpha \rceil}$. In the absence of such an event, $\mathsf{H}(\mathbb{S}_{\mathbb{C}}(x))$ would information theoretically appear random to $\mathcal{A}^{\mathsf{H}(\cdot)}$. Lemma 8 shows that such an event may only happen with negligible probability $q\epsilon$ where q is the total number of random oracle queries.

▶ **Lemma 8.** *For a some class $\mathbb{C}$ of pROM algorithms with $\delta-$safe function $\mathbb{S}_{\mathbb{C}}\{0,1\}^n \rightarrow \{0,1\}^*$, let $\mathsf{bad}_{\mathcal{A}}$ be the event that on some input $x \in \{0,1\}^n$, $\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}$ queries the random oracle at $\alpha \circ \mathbb{S}_{\mathbb{C}}(x)$ for any $\alpha > 0$. Then $\Pr[\mathsf{bad}_{\mathcal{A}}] \leq q\delta$, where q is the number of oracle queries made by $\mathcal{A}^{\mathsf{H}(\cdot)}$.*

Proof. We prove the claim by a reduction argument. By way of contradiction, suppose there exists a $\mathcal{B}^{\mathsf{H}(\cdot)} \in \mathbb{C}$ such that on input string x, $\mathcal{B}^{\mathsf{H}(\cdot)}$ makes q queries to the random oracle $\mathsf{H}(\cdot)$ and $\Pr[\mathsf{bad}_{\mathcal{B}}] > q\epsilon$. We construct an adversary $\mathcal{A}^{\mathsf{H}(\cdot)}$ as follows: on input x, the adversary

- Simulates $\mathcal{B}^{\mathsf{H}(\cdot)}$ with input x
- Keeps track of all q queries by which $\mathcal{B}^{\mathsf{H}(\cdot)}$ queries the random oracle
- On termination of $\mathcal{B}^{\mathsf{H}(\cdot)}$, returns the suffix of length $|\mathbb{S}_{\mathbb{C}}(\mathrm{x})|$ from one of the q queries selected uniformly at random

However, we know that $\mathcal{B}^{\mathsf{H}(\cdot)}$ queries the random oracle at $\alpha \circ \mathbb{S}_{\mathbb{C}}(x)$ with probability $> q\delta$. Since $\mathcal{A}^{\mathsf{H}(\cdot)}$ picks one of $\mathcal{B}^{\mathsf{H}(\cdot)}$'s queries at random, $\Pr[\mathcal{A}^{\mathsf{H}(\cdot)}(x) = \mathbb{S}_{\mathbb{C}}(x)] > \delta$, which contradicts the definition of $\delta-$safe function. ◀

Assuming that $\mathcal{A}^{\mathsf{H}(\cdot)}$never queries the random oracle at any point of the form $y \circ \mathbb{S}_{\mathbb{C}}(x)$ with $y \in \{0,1\}^{\lceil \log_2 \alpha \rceil}$ (for some $\alpha > 0$) we can view each $\mathsf{H}(y \circ \mathbb{S}_{\mathbb{C}}(x))$ as a fresh w-bit string. Thus, we can obtain a random $w\alpha$-bit string by concatenating all of the labels $\mathsf{H}(y \circ \mathbb{S}_{\mathbb{C}}(x))$ for each $y \in \{0,1\}^{\lceil \log_2 \alpha \rceil}$. This motivates the following definition of an *expansion family* which will be used in subsequent sections.

▶ **Definition 9** (Expansion Family). *For random oracle $\mathsf{H}(\cdot)$ the expansion family of functions $\{\mathsf{E}_\alpha^{\mathsf{H}(\cdot)}\}_{\alpha=1}^{\infty}$ where each function $\mathsf{E}_\alpha^{\mathsf{H}(\cdot)} : \{0,1\}^* \rightarrow \{0,1\}^{\alpha w}$ is defined as $\mathsf{E}_\alpha^{\mathsf{H}(\cdot)}(x) = \mathsf{H}(1 \circ x) \circ \mathsf{H}(2 \circ x) \circ \cdots \circ \mathsf{H}(\alpha \circ x)$, where the prefix $i \in [\alpha]$ of x for each oracle query in the definition is expressed in binary using $\lceil \log_2 \alpha \rceil$ bits.*

3.2 Framework for LDCs against Resource Bounded Channels

Our aim in this section is to achieve LDCs having no asymptotic loss in rate, query complexity, or success probability of private locally decodable codes. In contrast to the private LDC setting, we assume no private (or public) key setup assumptions. We also aim for LDCs that may be used for multiple (polynomial) rounds of communication, a notion which we describe later in the section. Let $\mathsf{C}_{\mathsf{ldc}^*}$ be a LDC*and $\mathsf{C}_{\mathsf{priv}}$ be a private-LDC. Then, against classes of pROM algorithms permitting $\delta-$safe functions, our encoder will use $\mathsf{C}_{\mathsf{ldc}^*}$ to bootstrap off of $\mathsf{C}_{\mathsf{priv}}$ even in the absence of shared private randomness with the decoder.

Framework Overview: The encoding algorithm first samples a random seed r of modest length (k_{ldc^*}). By embedding an encoding of r (via $\mathsf{C}_{\mathsf{ldc}^*}$) in our final codeword, we can ensure that our decoder will also have access to r. Let the channel, over which the communication happens, belong to a class $\mathbb{C}$ of pROM algorithms (w.r.t. random oracle $\mathsf{H}(\cdot)$) permitting some δ-safe function $\mathbb{S}_{\mathbb{C}} : \{0,1\}^{k_{\mathsf{ldc}^*}} \rightarrow \{0,1\}^*$. Even though the channel has access to the seed r, it will be unable to compute $\mathbb{S}_{\mathbb{C}}(\mathsf{r})$ by definition of the safe function. Thus $\mathsf{H}(\mathbb{S}_{\mathbb{C}}(\mathsf{r}))$ is effectively a random string to the channel. We can expand this randomness via an expansion function (Definition 9), and use $\mathsf{GenKey}_{\mathsf{priv}}$ with this randomness to compute a key. The computed key is effectively secret from the channel and can be used in conjunction with $\mathsf{Enc}_{\mathsf{priv}}$ to obtain an encoding of any input message. Note that since the decoder also has access to r, it may also compute the secret key using exactly the same procedure and use this key in conjunction with $\mathsf{Dec}_{\mathsf{priv}}$ to perform the required decoding. Thus the use of $\mathsf{C}_{\mathsf{ldc}^*}$, safe and expansion functions on a random seed reduces the setting to that of $\mathsf{C}_{\mathsf{priv}}$. Our framework is parameterized by $[\mathbb{S}_{\mathbb{C}}, \mathsf{C}_{\mathsf{ldc}^*}, \mathsf{C}_{\mathsf{priv}}]$.

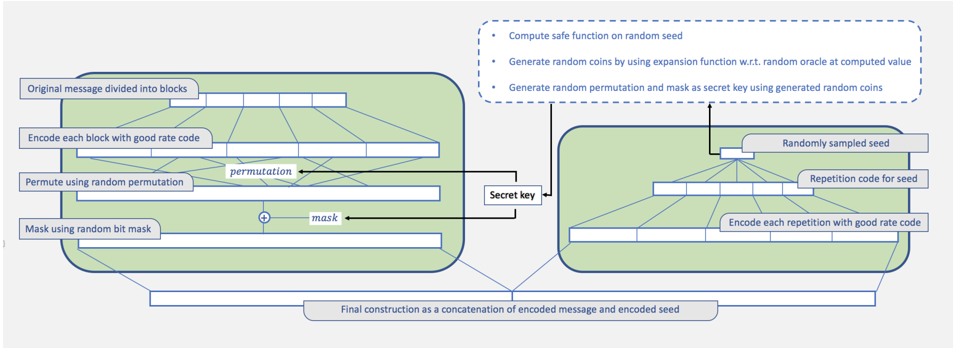

Figure 3 Instantiation of framework for LDCs against adversaries permitting safe functions.

Explicit Constructions: We provide explicit constructions of LDCs against adversarial pROM channels permitting $\delta-$safe functions by instantiating the framework discussed above. Figure 3 gives an overview of the instantiation. For private LDCs we use the constructions of Theorem 5. Furthermore, we instantiate $\mathsf{C}_{\mathsf{ldc}^*}$ as follows: The encoder encodes the seed with a standard constant rate error correcting code – we instantiate this with Justesen codes – composed with a repetition code. The local decoder then randomly samples seed-encodings and takes a majority vote over the decoded samples to determine the seed. We show that these encoding and decoding algorithms result in a (ℓ, ρ, p)-LDC* where, for a parameter α and message of length k, $\ell = \mathcal{O}(\alpha k)$, $\rho = \mathcal{O}(1)$ and $p \geq 1 - e^{-\alpha}$. The prior results are obtainable for any desired codeword length $K = \Omega(k)$. We refer the reader to the full version [13] for a formal explanation of this LDC* instantiation.

Detailed descriptions of our encoder ($\mathsf{Enc}_{\mathsf{final}}^{\mathsf{H}(\cdot)}$) and decoder ($\mathsf{Dec}_{\mathsf{final}}^{\mathsf{H}(\cdot)}$), given a message x, security parameter κ, and random oracle $\mathsf{H}(\cdot)$, may be described in Figure 4. In particular, our framework lead to the following theorem.

$\mathsf{Enc}_{\mathsf{final}}^{\mathsf{H}(\cdot)}(\mathbf{x}, \kappa)[\mathbb{S}_{\mathbb{C}}, \mathsf{C}_{\mathsf{ldc}^*}, \mathsf{C}_{\mathsf{final}}]$:

1. Sample a random seed of length k_{ldc^*}.
   $\mathsf{r} \leftarrow \{0,1\}^{k_{\mathsf{ldc}^*}}$
2. Encode random seed using an LDC.
   $Y_{\mathsf{ldc}^*} := \mathsf{Enc}_{\mathsf{ldc}^*}(\mathsf{r})$
3. Generate randomness uncomputable by channel via safe and expansion functions.
   $R := \mathsf{E}_{\tau}^{\mathsf{H}(\cdot)}(\mathbb{S}_{\mathbb{C}}(\mathsf{r}))$
4. Generate a secret key from the randomness.
   $\mathsf{sk}_{\mathsf{final}} := \mathsf{GenKey}_{\mathsf{priv}}(\kappa; R)$
5. Use private LDC encoder with generated key.
   $Y_{\mathsf{priv}} := \mathsf{Enc}_{\mathsf{priv}}(x, \kappa, \mathsf{sk}_{\mathsf{final}})$
6. **Output** $Y_{\mathsf{priv}} \circ Y_{\mathsf{ldc}^*}$

$\mathsf{Dec}_{\mathsf{final}}^{\mathsf{H}(\cdot), \mathbf{Y}'_{\mathsf{priv}} \circ \mathbf{Y}'_{\mathsf{ldc}^*}}(\mathbf{i}, \kappa)[\mathbb{S}_{\mathbb{C}}, \mathsf{C}_{\mathsf{ldc}^*}, \mathsf{C}_{\mathsf{final}}]$:

1. Decode the original random seed.
   $\mathsf{r} := \mathsf{Dec}_{\mathsf{ldc}^*}^{Y'_{\mathsf{ldc}^*}}(\cdot)$
2. Compute randomness used by encoder.
   $R := \mathsf{E}_{\tau}^{\mathsf{H}(\cdot)}(\mathbb{S}_{\mathbb{C}}(\mathsf{r}))$
3. Compute secret key used by encoder.
   $\mathsf{sk}_{\mathsf{final}} := \mathsf{GenKey}_{\mathsf{OPS}}(\kappa; R)$
4. Use private LDC decoder with computed key.
   Output $\mathsf{Dec}_{\mathsf{priv}}^{Y'_{\mathsf{priv}}}(i, \mathsf{sk}_{\mathsf{final}})$

Figure 4 Encoding and decoding algorithms for our LDC construction.

▶ **Theorem 10.** *Let* $\mathsf{C}_{\mathsf{ldc}^*}[K_{\mathsf{ldc}^*}, k_{\mathsf{ldc}^*}] = (\mathsf{Enc}_{\mathsf{ldc}^*}, \mathsf{Dec}_{\mathsf{ldc}^*})$ *be an* $(\ell_{\mathsf{ldc}^*}, \rho_{\mathsf{ldc}^*}, p_{\mathsf{ldc}^*})-\mathsf{LDC}^*$ *and* $\mathsf{C}_{\mathsf{priv}}[K_{\mathsf{priv}}, k_{\mathsf{priv}}, \kappa] = (\mathsf{Enc}_{\mathsf{priv}}, \mathsf{Dec}_{\mathsf{priv}}, \mathsf{GenKey}_{\mathsf{priv}})$ *be an* $(\ell_{\mathsf{priv}}, \rho_{\mathsf{priv}}, p_{\mathsf{priv}}, \epsilon_{\mathsf{priv}})-$*private* LDC. *Then for any class* $\mathbb{C}$ *of* pROM *algorithms admitting a* $\delta-$*safe function* $\mathbb{S}_{\mathbb{C}} : \{0,1\}^{k_{\mathsf{ldc}^*}} \to \{0,1\}^*$, *the* $(K_{\mathsf{final}}, k_{\mathsf{final}})_2$ *coding scheme in the random oracle model* $\mathsf{C}_{\mathsf{final}}[\mathbb{S}_{\mathbb{C}}, \mathsf{C}_{\mathsf{ldc}^*}, \mathsf{C}_{\mathsf{priv}}] = (\mathsf{Enc}^{\mathsf{H}(\cdot)}_{\mathsf{final}}, \mathsf{Dec}^{\mathsf{H}(\cdot)}_{\mathsf{final}})$ *is an* $(\ell_{\mathsf{final}}, \rho_{\mathsf{final}}, p_{\mathsf{final}}, \epsilon_{\mathsf{final}})$-LDC *with* $k_{\mathsf{final}} = k_{\mathsf{priv}}$, $K_{\mathsf{final}} = K_{\mathsf{ldc}^*} + K_{\mathsf{priv}}$, $\ell_{\mathsf{final}} = \ell_{\mathsf{final}} + \ell_{\mathsf{ldc}^*}$, $\rho_{\mathsf{final}} = \frac{1}{K_{\mathsf{ldc}^*} + K_{\mathsf{priv}}} \min\{\rho_{\mathsf{ldc}^*} K_{\mathsf{ldc}^*}, \rho_{\mathsf{priv}} K_{\mathsf{priv}}\}$, $p_{\mathsf{final}} \geq 1 - (2 - p_{\mathsf{priv}} - p_{\mathsf{ldc}^*})$, $\epsilon_{\mathsf{final}} \leq \epsilon_{\mathsf{priv}} + q\delta$. *Here* q *is an upper bound on the number of queries any algorithm* $\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}$ *makes to the random oracle* $\mathsf{H}(\cdot)$.

The final codeword generated by $\mathsf{Enc}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$ is simply the concatenation of the codewords generated by $\mathsf{Enc}_{\mathsf{priv}}$ and $\mathsf{Enc}_{\mathsf{ldc}^*}$, resulting in $K_{\mathsf{final}} = K_{\mathsf{ldc}^*} + K_{\mathsf{priv}}$. By construction, the only queries $\mathsf{Dec}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$ makes to the corrupted codeword are during the executions of $\mathsf{Dec}_{\mathsf{ldc}^*}$ and $\mathsf{Dec}_{\mathsf{priv}}$. This gives the locality $\ell_{\mathsf{final}} = \ell_{\mathsf{ldc}^*} + \ell_{\mathsf{priv}}$. Furthermore for correct overall decoding, it is necessary that the individual codes are correctly decoded. Thus the total errors that the code can tolerate is bounded by the maximum number of errors any individual one of the codes can tolerate. This gives the claimed (worst case) error rate. We emphasize that the proofs of the bounds on the decoder's success probability and the security of the framework is much more involved than the above discussion and is included in Section 3.3 and 3.4. In particular, we show that no adversary admitting δ-safe functions can distinguish between the encodings of $\mathsf{Enc}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$ and those of $\mathsf{Enc}_{\mathsf{priv}}$ with random strings appended to them. Furthermore, even the decoder, who has no computational restrictions and gets the appropriate secret key used during the respective encoding processes may not make this distinction, thereby effectively reducing the security of $\mathsf{C}_{\mathsf{final}}$ to that of $\mathsf{C}_{\mathsf{priv}}$ with negligible loss. The following two corollaries exhibit decoding probability vs locality tradeoffs when our framework is instantiated with LDC^*s and private-LDCs from the discussion of explicit constructions earlier in the section. We defer details of these to the full version [13].

▶ **Corollary 11.** *For security parameter* κ, *a class* $\mathbb{C}$ *of* pROM *adversaries admitting* $\delta-$*safe function* $\mathbb{S}_{\mathbb{C}} : \{0,1\}^{\log^{1+\varepsilon}\kappa} \to \{0,1\}^*$ *where* $\varepsilon > 0$ *and for every* $k > 0$ *such that* $k = \mathrm{poly}(\kappa)$ *where* poly *is any non-zero polynomial, there exists a* $(\beta k, k)_2$ *coding scheme in the random oracle model that is an* $(\ell, \rho, p, \epsilon, \mathbb{C})-\mathsf{LDC}$ *where* $\ell = (\alpha+1)\log^{1+\varepsilon}\kappa$ *(such that* $\alpha \geq 17$*),* ρ *is a constant,* p *is a constant dependent on* α, *and* $\epsilon \leq \mathsf{negl}(\kappa) + q\delta$. *Here* β *is a constant,* $\mathsf{negl}(\kappa)$ *is a negligible function of* κ *and* q *is an upper bound on the total queries any algorithm in* $\mathbb{C}$ *makes to the random oracle.*

▶ **Corollary 12.** *For security parameter* κ, *a class* $\mathbb{C}$ *of* pROM *adversaries admitting* $\delta-$*safe function* $\mathbb{S}_{\mathbb{C}} : \{0,1\}^{\log^{1+\varepsilon}\kappa} \to \{0,1\}^*$ *where* $\varepsilon > 0$ *and for every* $k > 0$ *such that* $k = \mathrm{poly}(\kappa)$ *where* poly *is any non-zero polynomial, there exists a* $(\beta k, k)_2$ *coding scheme in the random oracle model that is an* $(\ell, \rho, p, \epsilon, \mathbb{C})-\mathsf{LDC}$ *where* $\ell = (1 + 24\log^{1+\varepsilon}\kappa)\log^{1+\varepsilon}\kappa$, ρ *is a constant,* $p \geq (1 - \mathsf{negl}_1(\kappa))$, *and* $\epsilon \leq \mathsf{negl}_2(\kappa) + q\delta$. *Here* β *is a constant,* $\mathsf{negl}_1(\kappa)$ *and* $\mathsf{negl}_2(\kappa)$ *are negligible functions of* κ, *and* q *is an upper bound on the total queries any algorithm in* $\mathbb{C}$ *makes to the random oracle.*

Precomputation. We remark that Steps 1-4 of $\mathsf{Enc}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$ may be precomputed. This may be advantageous in some settings to speed up encoding time as the sender may precompute multiple $(\mathsf{sk}_{\mathsf{final}}, Y_{\mathsf{ldc}^*})$ pairs. When a message is ready to be encoded, the sender then simply needs to generate Y_{priv} using $\mathsf{sk}_{\mathsf{final}}$ and append Y_{ldc^*} to generate the final codeword. However, we do note that this precomputation must be done after the selection of the random oracle, and that such precomputation is not possible for $\mathsf{Dec}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$.

Multi-round Communication. Existing constructions of private LDCs [33] are secure only for a single *round* of communication (see the full version [13] for details on the round-based game between the encoder/decoder and the channel in the private LDC setting). We may generalize our model to be in terms of rounds as well, where each round runs an instance of the experiment $\texttt{LDC} - \texttt{Sec} - \texttt{Game}$ defined in Section 2.4. We remark that our codes work for this generalized model as well. In every round of the experiment, the encoder can sample a fresh random seed r. This is not directly possible in the existing private LDC constructions as an attacker listening to the decoder's queries may learn information about the secret key after a single round of communication. For this Ostrovsky et al. introduce a new construction which hides the secret key behind a layer of encryption, which in turn increases the locality of their final constructions to $\omega(\log^2 \kappa)$.

3.3 Two-Phase Hybrid Distinguisher Argument

To prove the security of the LDC framework in section 3.2, our approach is to argue the following: if any channel wins the `LDC-Sec-Game` against an instantiation of our LDC constructions $(\mathsf{Enc}^{\mathsf{H}(\cdot)}_{\mathsf{final}}, \mathsf{Dec}^{\mathsf{H}(\cdot)}_{\mathsf{final}})$, then this channel can win the `priv-LDC-Sec-Game` against its constituent private-LDC (contradicting its security guarantee).

Standard Hybrid Argument Failure: A natural attempt to prove this, yet one that fails, is to use the following standard hybrid argument. In the first hybrid we use our original encoding scheme $\mathsf{Enc}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$ to obtain a codeword ${\mathsf{Y}_{\mathsf{priv}}}^{(0)} \circ {\mathsf{Y}_{\mathsf{ldc}}}^{(0)}$. In the second hybrid, we replace the second component with an encoding of a random unrelated nonce to get ${\mathsf{Y}_{\mathsf{priv}}}^{(1)} \circ {\mathsf{Y}_{\mathsf{ldc}^*}}^{(1)}$. Here ${\mathsf{Y}_{\mathsf{ldc}^*}}^{(1)}$ is an encoding of some random nonce which is sampled completely independent of the message encoding ${\mathsf{Y}_{\mathsf{priv}}}^{(1)}$. We would like to argue that the two hybrids are indistinguishable and conclude that a resource bounded channel cannot fool the local decoder from original encoding scheme (first hybrid) – since we cannot fool the private-LDC local decoder in the second hybrid. However, if the distinguisher $\mathcal{D}$ is able to evaluate the safe-function then the hybrids are trivially distinguishably. On the other hand, if we assume that the distinguisher $\mathcal{D}$ is resource bounded like the channel then indistinguishability does not suffice to argue that the local decoder i.e., fooling the decoder does not yield a resource bounded distinguisher $\mathcal{D}$ since the decoder is not constrained in the same way as the resource bounded channel.

Two-Phase Argument Overview: We address the previous issue by introducing a *two-phase distinguisher game* defined over adversary/distinguisher pairs. In the first phase of this game, a random coin toss $b \in \{0, 1\}$ randomly selects one of the hybrid encoders to encode a message. The selected hybrid hands its encoding ${\mathsf{Y}_{\mathsf{priv}}}^{(b)} \circ {\mathsf{Y}_{\mathsf{ldc}^*}}^{(b)}$ to the adversary $\mathcal{A}^{\mathsf{H}(\cdot)}$ which outputs a corrupted codeword ${\mathsf{Y}_{\mathsf{hyb}}}^{(b)'}$. In the second phase, the distinguisher $\mathcal{D}$ is given the initial message x, the corrupted codeword ${\mathsf{Y}_{\mathsf{hyb}}}^{(b)'}$, along with the secret key $\mathsf{sk}^{(b)}$ used to obtain ${\mathsf{Y}_{\mathsf{priv}}}^{(b)}$, and tries to predict the value of b, i.e., which hybrid encoder was used. An important point to note is that $\mathcal{D}$ is not constrained in any way. However, it is not given access to the random oracle. We show (Lemma 14) that for any such attacker-distinguisher pair, the distinguisher succeeds at guessing which hybrid encoding was used with at most negligible probability. The two phase hybrid argument allows us to reason about our original goal: the probability that the channel fools the honest decoder. In particular, a channel that wins the `LDC-Sec-Game` with non-negligible probability can be used in phase 1 in conjunction with a distinguisher that can simulate the decoding algorithm (with the correct key) in phase 2 to distinguish between the hybrids with non-negligible probability. This gives the required contradiction (Lemma 15). We formally define the two hybrid encoders in Figure 5.

$\mathsf{Enc}_\mathbf{0}^{\mathsf{H}(\cdot)}(\mathbf{x},\kappa)[\mathbb{S}_\mathbb{C},\mathsf{C}_{\mathsf{ldc}^*},\mathsf{C}_{\mathsf{priv}}]$: **(same as Figure 3)**	$\mathsf{Enc}_\mathbf{1}(\mathbf{x},\mathsf{sk}^{(\mathbf{1})},\kappa)[\mathbb{S}_\mathbb{C},\mathsf{C}_{\mathsf{ldc}^*},\mathsf{C}_{\mathsf{priv}}]$:
1. Sample a random seed of length k_{ldc^*}. $\mathsf{r}^{(0)} \leftarrow \{0,1\}^{k_{\mathsf{ldc}^*}}$ **2.** Encode random seed using an LDC*. ${\mathsf{Y}_{\mathsf{ldc}}}^{(0)} := \mathsf{Enc}_{\mathsf{ldc}^*}(\mathsf{r}^{(0)})$ **3.** Generate randomness uncomputable by channel via safe and expansion functions. $\mathsf{R}^{(0)} := \mathsf{E}_\tau^{\mathsf{H}(\cdot)}(\mathbb{S}_\mathbb{C}(\mathsf{r}^{(0)}))$ **4.** Generate a secret key from the randomness. $\mathsf{sk}^{(0)} := \mathsf{GenKey}_{\mathsf{priv}}(\kappa;\mathsf{R}^{(0)})$ **5.** Use private LDC encoder with generated key. ${\mathsf{Y}_{\mathsf{priv}}}^{(0)} := \mathsf{Enc}_{\mathsf{priv}}(x,\kappa,\mathsf{sk}^{(0)})$ **6. Output** ${\mathsf{Y}_{\mathsf{priv}}}^{(0)} \circ {\mathsf{Y}_{\mathsf{ldc}}}^{(0)}$	**1.** Sample a random seed of length k_{ldc^*}. $\mathsf{r}^{(1)} \leftarrow \{0,1\}^{k_{\mathsf{ldc}^*}}$ **2.** Encode random seed using an LDC*. ${\mathsf{Y}_{\mathsf{ldc}^*}}^{(1)} := \mathsf{Enc}_{\mathsf{ldc}^*}(\mathsf{r}^{(1)})$ **3.** Use private LDC encoder with input key. ${\mathsf{Y}_{\mathsf{priv}}}^{(1)} := \mathsf{Enc}_{\mathsf{priv}}(x,\kappa,\mathsf{sk}^{(1)})$ **4. Output** ${\mathsf{Y}_{\mathsf{priv}}}^{(1)} \circ {\mathsf{Y}_{\mathsf{ldc}^*}}^{(1)}$

Figure 5 Hybrid encoding algorithms. By design, $\mathsf{Enc}_0^{\mathsf{H}(\cdot)}$ is the same as our proposed LDC construction.

Let $\mathcal{A}^{\mathsf{H}(\cdot)}$ be an adversarial channel belonging to a class $\mathbb{C}$ of pROM algorithms w.r.t random oracle $\mathsf{H}(\cdot)$ permitting $\delta-$safe functions. Furthermore, let $\mathcal{D} : \left(\{0,1\}^*\right)^4 \to \{0,1\}$ be a computationally unbounded algorithm. We will term $\mathcal{A}^{\mathsf{H}(\cdot)}$ and $\mathcal{D}$ as *attacker* and *distinguisher* respectively. Using the *hybrid* encoders in Figure 5, we define the *indistinguishability experiment* $\mathsf{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},\kappa,x}$ over all *attacker-distinguisher pairs* $(\mathcal{A}^{\mathsf{H}(\cdot)},\mathcal{D})$. Note that in this experiment, $\mathcal{D}$ is provided with the secret key that the selected hybrid used during encoding, and does not have access to the random oracle. With respect to this experiment, we define the *advantage* of the attacker-distinguisher pair as follows:

$$\mathsf{Adv}_{\mathcal{A},\mathcal{D}} := \max_x \left| \Pr[\mathsf{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},\kappa,x} = 1] - \frac{1}{2} \right|$$

where the probability is taken over the randomness of $\mathcal{D}$, $\mathcal{A}^{\mathsf{H}(\cdot)}$, and the selection of the random oracle $\mathsf{H}(\cdot)$. Our first aim will be to show that the advantage of any attacker-distinguisher pair, as defined above, is negligible at best.

Let $(\mathcal{A}^{\mathsf{H}(\cdot)},\mathcal{D})$ be any attacker-distinguisher pair and hybrid encoders be instantiated with parameters $[\mathbb{S}_\mathbb{C},\mathsf{C}_{\mathsf{ldc}^*},\mathsf{C}_{\mathsf{priv}}]$. For security parameter κ and message x, consider an execution of the indistinguishability experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},\kappa,x}$. Let $\mathsf{bad}_\mathcal{A}$ be the event that the attacker queries the random oracle at position $c \circ \mathbb{S}_\mathbb{C}(\mathsf{r}^{(b)})$ where r is the random seed chosen by the selected hybrid encoder $\mathsf{Enc}_b^{\mathsf{H}(\cdot)}$ and c is any constant expressed in binary. Furthermore, let succ be the event where the attacker-distinguisher pair succeed in distinguishing the hybrid encodings in the experiment, i.e., the event where $\mathsf{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},\kappa,x} = 1$.

The next proposition follows from the observation that conditioning on the event $\mathsf{bad}_\mathcal{A}$ not occurring, the secret key sk_b used during the encoding process remains (information theoretically) private to both the adversary and the distinguisher. To the pair, $\mathsf{Enc}_0^{\mathsf{H}(\cdot)}$ appears information theoretically identical to Enc_1 which gets a secret key as its input, and thus any advantage on distinguishing the encoding schemes would allow the pair to distinguish between random strings.

▶ **Proposition 13.** $\Pr[\mathsf{succ}|\overline{\mathsf{bad}}_\mathcal{A}] = 1/2$

$\texttt{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},\kappa,x}$: \\message x and security parameter κ:

Phase I

1. Encode message with both hybrids. Let $\mathsf{sk}^{(0)}$ and $\mathsf{sk}^{(1)}$ be the secret keys used by first and second hybrid respectively.
   $\mathsf{Y_{hyb}}^{(0)} := \mathsf{Enc}_0^{\mathsf{H}(\cdot)}(x,\kappa)$.
   $\mathsf{Y_{hyb}}^{(1)} := \mathsf{Enc}_1(x,\kappa,\mathsf{sk}^{(1)})$.
2. Flip an unbiased coin to randomly select a hybrid encoding.
   $b \leftarrow \{0,1\}$
3. Hand the selected encoding to the channel to get corrupted codeword.
   $\mathsf{Y_{hyb}}^{(b)'} := \mathcal{A}^{\mathsf{H}(\cdot)}(x,\kappa,\mathsf{Y_{hyb}}^{(b)})$

Phase II

1. Distinguisher, given the message, secret key, corrupted codeword, and security parameter, guesses the coin toss.
   $b' := \mathcal{D}(x,\mathsf{sk}^{(b)},\mathsf{Y_{hyb}}^{(b)'},\kappa)$
2. $\texttt{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},x,\kappa} = \begin{cases} 1 & \text{iff } b' = b \\ 0 & \text{otherwise} \end{cases}$

Figure 6 Indistinguishability experiment for the attacker-distinguisher pair.

The following lemma shows that the advantage for any attacker-distinguisher pair is negligible.

▶ **Lemma 14.** $\mathsf{Adv}_{\mathcal{A},\mathcal{D}} \le \frac{q\delta}{2}$ *for any execution of the game* $\texttt{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},x,\kappa}$. *Here q is an upper bound on the number of queries $\mathcal{A}^{\mathsf{H}(\cdot)}$ makes to the random oracle.*

Proof. Consider some execution of the game $\texttt{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},x,\kappa}$. Using conditional probability to partition the event space, the advantage of the attacker-distinguisher pair is:

$$\mathsf{Adv}_{\mathcal{A},\mathcal{D}} = \left|\Pr[\mathsf{succ}] - \frac{1}{2}\right| = \left|\Pr[\mathsf{succ}|\mathsf{bad}_{\mathcal{A}}]\Pr[\mathsf{bad}_{\mathcal{A}}] + \Pr[\mathsf{succ}|\overline{\mathsf{bad}}_{\mathcal{A}}]\Pr[\overline{\mathsf{bad}}_{\mathcal{A}}] - \frac{1}{2}\right|$$

By Proposition 13, we may view the event of succ conditioned on $\mathsf{bad}_{\mathcal{A}}$ not occurring as an unbiased random choice. Thus $\mathsf{Adv}_{\mathcal{A},\mathcal{D}} = \left|\Pr[\mathsf{succ}|\mathsf{bad}_{\mathcal{A}}]\Pr[\mathsf{bad}_{\mathcal{A}}] + \frac{1}{2}(1-\Pr[\mathsf{bad}_{\mathcal{A}}]) - \frac{1}{2}\right|$. This allows us to bound the advantage of the attacker-distinguisher pair by a factor of the probability of event bad occurring by $\mathsf{Adv}_{\mathcal{A},\mathcal{D}} = \Pr[\mathsf{bad}_{\mathcal{A}}]\left|\Pr[\mathsf{succ}|\mathsf{bad}_{\mathcal{A}}] - \frac{1}{2}\right| \le \Pr[\mathsf{bad}_{\mathcal{A}}]\frac{1}{2}$. Therefore by Lemma 8, the advantage of the attacker-distinguisher pair for the execution of $\texttt{Exp}_{\mathcal{A},\mathcal{D},\mathsf{H},x,\kappa}$ is at most $\frac{q\delta}{2}$. ◀

3.4 Security and Decoding Probability of Constructions

Note that $\mathsf{Enc}_0^{\mathsf{H}(\cdot)}$ is identical to $\mathsf{Enc}_{\mathsf{final}}^{\mathsf{H}(\cdot)}$ and Enc_1 is identical to $\mathsf{Enc}_{\mathsf{priv}}$ with random strings appended to its output. Consider a $(\ell_{\mathsf{priv}},\rho_{\mathsf{priv}},p_{\mathsf{priv}},\epsilon_{\mathsf{priv}})-$private LDC instance $\mathsf{C}_{\mathsf{priv}}[k_{\mathsf{priv}},K_{\mathsf{priv}}] = (\mathsf{Enc}_{\mathsf{priv}},\mathsf{Dec}_{\mathsf{priv}},\mathsf{GenKey}_{\mathsf{priv}})$ and an instantiation of our constructions $\mathsf{C}_{\mathsf{final}}[\mathbb{S}_{\mathbb{C}},\mathsf{C}_{\mathsf{ldc}^*},\mathsf{C}_{\mathsf{final}}] = (\mathsf{Enc}_{\mathsf{final}}^{\mathsf{H}(\cdot)},\mathsf{Dec}_{\mathsf{final}}^{\mathsf{H}(\cdot)})$. With respect to these instances, we define $\epsilon_{\mathsf{final}}$ as the following:

$$\epsilon_{\mathsf{final}} := \Pr[\texttt{LDC}-\texttt{Sec}-\texttt{Game}[\mathcal{A},x,\mathsf{H},\kappa,\rho,p] = 1 \text{ against } \mathsf{C}_{\mathsf{final}}]$$

Consider the codes $\mathsf{C}_0 = (\mathsf{Enc}_0^{\mathsf{H}(\cdot)},\mathsf{Dec}_{\mathsf{final}}^{\mathsf{H}(\cdot)})$ and $\mathsf{C}_1 = (\mathsf{Enc}_1,\mathsf{Dec}_{\mathsf{priv}^*})$ formed by our hybrid encoders. Here $\mathsf{Dec}_{\mathsf{priv}^*}$ is defined identical to $\mathsf{Dec}_{\mathsf{priv}}$ except that it ignores the strings appended to the output of $\mathsf{Enc}_{\mathsf{priv}}$ during the encoding execution of Enc_1. With respect to these codes, we define the following:

$$\epsilon_0 := \max_{\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}} \Pr[\mathtt{priv-LDC-Sec-Game}[\mathcal{A}, x, \kappa, \rho_{\mathsf{final}}, p_{\mathsf{final}}] = 1 \text{ against } \mathsf{C}_0]$$

$$\epsilon_1 := \max_{\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}} \Pr[\mathtt{priv-LDC-Sec-Game}[\mathcal{A}, x, \kappa, \rho_{\mathsf{final}}, p_{\mathsf{final}}] = 1 \text{ against } \mathsf{C}_1]$$

Note that by our definitions, $\epsilon_0 = \epsilon_{\mathsf{final}}$ and $\epsilon_1 \le \epsilon_{\mathsf{priv}}$. The second observation follows from the following:

$$\begin{aligned}\epsilon_1 &= \max_{\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}} \Pr[\mathtt{priv-LDC-Sec-Game}[\mathcal{A}, x, \kappa, \rho_{\mathsf{final}}, p_{\mathsf{final}}] = 1 \text{ against } \mathsf{C}_1] \\ &\le \max_{\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}} \Pr[\mathtt{priv-LDC-Sec-Game}[\mathcal{A}, x, \kappa, \rho_{\mathsf{final}}, p_{\mathsf{priv}}] = 1 \text{ against } \mathsf{C}_1] \\ &\le \max_{\mathcal{A} \in \mathbb{C}} \Pr[\mathtt{priv-LDC-Sec-Game}[\mathcal{A}, x, \kappa, \rho_{\mathsf{priv}}, p_{\mathsf{priv}}] = 1 \text{ against } \mathsf{C}_{\mathsf{priv}}] = \epsilon_{\mathsf{priv}}\end{aligned}$$

where the first inequality follows because $p_{\mathsf{final}} \le p_{\mathsf{priv}}$, while the second inequality follows since $\rho_{\mathsf{final}} K_{\mathsf{final}} \le \rho_{\mathsf{priv}} K_{\mathsf{priv}}$ i.e., the attacker gets to make more corruptions against $\mathsf{C}_{\mathsf{priv}}$. Lemma 15 upper bounds $|\epsilon_0 - \epsilon_1| \le q\delta$ and it immediately follows that $\epsilon_{\mathsf{final}} \le \epsilon_{\mathsf{priv}} + q\delta$.

▶ **Lemma 15.** $|\epsilon_0 - \epsilon_1| \le q\delta$. *Here q is an upper bound on the number of queries the attacker makes to the random oracle.*

Proof. Recall that an attacker wins the $\mathtt{LDC-Sec-Game}[\mathcal{A}, x, \mathsf{H}, \kappa, \rho, p]$ if there exists some index which the corresponding decoder fails to decode with probability at least p. Suppose for sake of contradiction that $|\epsilon_0 - \epsilon_1| > q\delta$ for some attacker $\mathcal{A}^{\mathsf{H}(\cdot)}$. Consider the distinguisher D' in Figure 7. With respect to the indistinguishability experiment, D' takes as input the original message x, the corrupted codeword y'_b, the key used by hybrid b during encoding, and the security parameter κ.

Distinguisher $\mathsf{D}'(x, y'_b, \mathsf{sk}_b, \kappa)$:

1. Computes ϵ_b by enumerating over all i, running $\mathsf{Dec}_{\mathsf{priv}}^{y'_b}(i, \kappa)$ and checking whether $\mathsf{Dec}_{\mathsf{priv}}$ fails to decode correctly with probability at least p_{priv}.
2. return $b' = \begin{cases} 1 & \text{with probability } \epsilon_b \\ 0 & \text{otherwise} \end{cases}$

Figure 7 Distinguisher that uses the $\mathsf{Dec}_{\mathsf{priv}}$ decoding algorithm.

Note that the computationally intensive step 1 of D' is possible since we assume no computational restrictions. Thus by conditional probability, the advantage of distinguisher D' paired with any $\mathcal{A}^{\mathsf{H}(\cdot)} \in \mathbb{C}$ may be given by

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A},\mathcal{D}'} &= \left|\Pr[\mathsf{succ}] - \frac{1}{2}\right| = \frac{1}{2}\left|\Pr[\mathsf{succ}|b=0] - \Pr[\overline{\mathsf{succ}}|b=1]\right| \\ &= \frac{1}{2}\left|(1-\epsilon_0) - (1-\epsilon_1)\right| = \frac{1}{2}|\epsilon_1 - \epsilon_0|,\end{aligned}$$

where the penultimate equality is by definition of the distinguisher D'. Our initial assumption $|\epsilon_0 - \epsilon_1| > q\delta$ then implies that $\mathsf{Adv}_{\mathcal{A},\mathcal{D}} > \frac{q\delta}{2}$, contradicting Lemma 14. ◀

The following proposition is a direct consequence of Lemma 15 and the observation that $\epsilon_1 \le \epsilon_{\mathsf{priv}}$.

▶ **Proposition 16.** $\epsilon_0 \leq \epsilon_{\mathsf{priv}} + q\delta$ *where q is an upper bound to the number of queries that the attacker makes to the random oracle.*

Finally, we complete the proof by showing that that $\epsilon_{\mathsf{final}} \leq \epsilon_0$ in Lemma 17. Combined with proposition 16 this completes the proof since $\epsilon_{\mathsf{final}} \leq \epsilon_0 + q\delta$.

▶ **Lemma 17.** $\epsilon_{\mathsf{final}} \leq \epsilon_0$

Proof. Let fail_i denote the event that $\mathsf{Dec}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$ incorrectly decodes x_i for $i \in [k]$. We define succ to be the event that $\mathtt{priv-LDC-Sec-Game}[\mathcal{A}, x, \kappa, \rho_{\mathsf{final}}, p_{\mathsf{final}}] = 1$ against C_0 to simplify notation. It suffices to argue that $\Pr[\mathsf{fail}_i|\overline{\mathsf{succ}}] \leq (1 - p_{\mathsf{priv}}) + (1 - p_{\mathsf{ldc}^*})$ for any $i \in [k]$ since $\Pr[\mathsf{succ}] = \epsilon_0$. Let key be the event that $\mathsf{Dec}^{\mathsf{H}(\cdot)}_{\mathsf{final}}$ recovers the correct seed $\mathsf{r}^{(0)}$ from ${\mathsf{Y}_{\mathsf{ldc}}}^{(0)}$. We first observe that

$$\begin{aligned}\Pr[\mathsf{fail}_i|\overline{\mathsf{succ}}] &= \Pr[\mathsf{fail}_i|\overline{\mathsf{succ}}, \mathsf{key}] \Pr[\mathsf{key}|\overline{\mathsf{succ}}] + \Pr[\mathsf{fail}_i|\overline{\mathsf{succ}}, \overline{\mathsf{key}}] \Pr[\overline{\mathsf{key}}|\overline{\mathsf{succ}}] \\ &\leq \Pr[\mathsf{fail}_i|\overline{\mathsf{succ}}, \mathsf{key}] + \Pr[\overline{\mathsf{key}}|\overline{\mathsf{succ}}]\end{aligned}$$

Second we observe that $\Pr[\overline{\mathsf{key}}|\overline{\mathsf{succ}}] \leq 1 - p_{\mathsf{ldc}^*}$ since there are at most $\rho_{\mathsf{final}} K_{\mathsf{final}} \leq \rho_{\mathsf{ldc}^*} K_{\mathsf{ldc}^*}$ errors in the second part of the codeword ${\mathsf{Y}_{\mathsf{ldc}}}^{(0)}$. Finally, observe that by definition we have $\Pr[\mathsf{fail}_i|\overline{\mathsf{succ}}, \mathsf{key}] \leq 1 - p_{\mathsf{priv}}$. The claim now directly follows. ◀

4 Constructing Safe Functions

In this section we provide several examples of safe functions in the parallel random oracle model (pROM) [6]. We first define the parallel random oracle model and introduce several cost metrics that measure the resources used by a pROM algorithm $\mathcal{A}^{\mathsf{H}(\cdot)}$.

4.1 Parallel Random Oracle Model

Computation in the pROM proceeds in rounds. Each round ends when the algorithm $\mathcal{A}$ outputs a batch of random oracle queries to be answered in parallel and a new round begins when the attacker receives the answer(s) to this batch of queries. In between rounds the $\mathcal{A}$ may perform arbitrary computation. Formally, in the initial round the pROM algorithm $\mathcal{A}$ takes input x, performs some arbitrary computation, and outputs a state σ_1 and list $\vec{u}_1 = (u^1_1, \ldots, u^1_{q_1})$ of random oracle queries. In general, we then have $(\vec{u}_{i+1}, \sigma_{i+1}) = \mathcal{A}(\sigma_i, \vec{a}_i)$ where $\vec{a}_i = (\mathsf{H}(u^i_1), \ldots, \mathsf{H}(u^i_{q_i})$ are the answers to the q_i random oracle queries $\vec{u}_i = (u^i_1, \ldots, u^i_{q_i})$ asked in the previous round. The execution ends in round t if the algorithm $\mathcal{A}$ returns an output value $y = \sigma_t$ along with an empty batch of random oracle queries $\vec{u}_t = \emptyset$. We use $\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x) = (\sigma_1, \sigma_2 \cdots, \sigma_t, \vec{u}_1, \ldots, \vec{u}_t)$ to denote the sequence of states (and oracle queries) output when we run the pROM attacker $\mathcal{A}(x)$ on input x fixing the random oracle $\mathsf{H}(\cdot)$ and fixing $\mathcal{A}$'s random coins R.

Cost Metrics. Figure 8 defines the *resources* we will consider as characterizing the cost of a particular execution trace $\mathcal{T} = \mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)$. We can define the time (resp. space) cost as $\mathsf{time}(\mathcal{T}) = t$ (resp. $\mathsf{space}(\mathcal{T}) = \max_{i \leq t} |\sigma_i|$). Similarly, the space time cost measures the product $\mathsf{space-time}(\mathcal{T}) = t \cdot \max_{i \leq t} |\sigma_i|$ and cumulative memory complexity measures $\mathsf{CMC}(\mathcal{T}) = \sum_{i=0}^{t} |\sigma_i|$. Intuitively, cumulative memory complexity captures the amortized space time complexity of a function that we want to evaluate many times in parallel [6]. Finally, the cumulative query cost is $\mathsf{CQ}(\mathcal{T}) = \sum_{i=1}^{t} |\vec{u}_i|$. For a resource $\mathcal{R}$ listed in Figure 8, the term $\mathcal{R}$ *complexity* will refer to a upper bound on resource $\mathcal{R}$.

Resource	Notation	Definition
Time	$\mathsf{time}(\mathcal{T})$	t
Space	$\mathsf{space}(\mathcal{T})$	$\max_{i=0}^{t} \lvert\sigma_i\rvert$
Space-Time	$\mathsf{ST}(\mathcal{T})$	$\mathsf{space}(\mathcal{T}) \cdot \mathsf{time}(\mathcal{T})$
Cumulative memory	$\mathsf{CMC}(\mathcal{T})$	$\sum_{i=0}^{t} \lvert\sigma_i\rvert$
Cumulative query	$\mathsf{CQ}(\mathcal{T})$	$\sum_{i=0}^{t} \vec{u}_i$

Figure 8 Resource Definitions.

▶ **Definition 18.** *(Resource Bounded Algorithms) We use $\mathcal{C}_{\mathsf{CQ},q}$ to refer to the set of all* pROM *algorithms $\mathcal{A}$ with the property that for all inputs x, random oracles $\mathsf{H}(\cdot)$, and all random strings R, we have $\mathsf{CQ}(\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)) \leq q$. We use $\mathcal{C}_{\mathsf{space},M,} \subset \mathcal{C}_{\mathsf{CQ},q}$ to refer to the subset of all* pROM *algorithms $\mathcal{A}$ with the additional constraint that for all inputs x, random oracles $\mathsf{H}(\cdot)$, and all random strings R, we have $\mathsf{CQ}(\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)) \leq q$ and $\mathsf{space}(\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)) \leq M$. Similarly, $\mathcal{C}_{\mathsf{time},T,q} \subset \mathcal{C}_{\mathsf{CQ},q}$ (resp. $\mathcal{C}_{\mathsf{space-time},S,q} \subset \mathcal{C}_{\mathsf{CQ},q}$) refers to the subset of all* pROM *algorithms $\mathcal{A}$ with the additional constraint that for all inputs x, random oracles $\mathsf{H}(\cdot)$, and all random strings R, we have $\mathsf{time}(\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)) \leq T$ (resp. $\mathsf{space-time}(\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)) \leq S$). The definition of $\mathcal{C}_{\mathsf{CMC},M,q}$ is symmetric – we add the additional constraint that $\mathsf{CMC}(\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)) \leq M$ for all $x, R, \mathsf{H}(\cdot)$.*

The assumption that the channel is resource constrained with respect to one or more of the above resources (time, space, cmc, etc.) is natural in most real word settings. For example, if a low latency channel uses $\mathcal{A}^{\mathsf{H}(\cdot)}$ to compute the corruptions to an encoded message then we can plausibly assume that the attacker $\mathcal{A} \in \mathcal{C}_{time,M,q}$ is time bounded – M denotes the maximum number of sequential evaluations of $\mathsf{H}(\cdot)$ before the corrupted codeword must be delivered. It would also be reasonable to assume that the total number of random oracle queries q is polynomial in the relevant parameters. One can also argue that in most practical settings the channel $\mathcal{A}$ will have other resource constraints e.g., space-bounded etc. In general one can define complexity classes for various combinations of resource constraints – see Definition 19.

▶ **Definition 19.** *For constraints $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_p)$ on resources $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_p)$ listed in Figure 8, the constraint class $C_{\mathcal{R},\mathcal{M}}$ is the set of all* pROM *$\mathcal{A}^{\mathsf{H}(\cdot)}$ such that $\mathcal{A}^{\mathsf{H}(\cdot)}$ is $\mathcal{R}$-bounded with constraints $\mathcal{M}$. Here, a* pROM *algorithm is said to be $\mathcal{R}$-*bounded *with* constraints *$\mathcal{M}$ if for all $i \leq p$ and on all inputs x, random coins R, and random oracles $\mathsf{H}(\cdot)$, we have*

$$\mathcal{R}_i\big(\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)\big) \leq \mathcal{M}_i$$

SCRYPT. Alwen et al. [5] proved that Percival's [35] memory hard function `scrypt` is maximally memory hard. In particular, $\mathtt{scrypt}_N$ can be computed in sequential time N, but any pROM attacker evaluating the scrypt function has cumulative memory complexity at least $\Omega(N^2 w)$, where w is the length of the output. Thus, `scrypt` could be used to obtain safe functions for the classes $\mathcal{C}_{\mathsf{CMC},S,q}$ and $\mathcal{C}_{\mathsf{space-time},S,q}$ – observe that $\mathsf{CMC}(\mathcal{T}) \leq \mathsf{space-time}(\mathcal{T})$ for any execution trace $\mathcal{T}$.

4.2 Sequentially Hard Function

The hash iteration function $f(x) = \mathsf{H}(x)^{t+1}$, defined recursively as $\mathsf{H}(x)^{t+1} = \mathsf{H}(\mathsf{H}(x)^t)$ where $\mathsf{H}(x)^1 = \mathsf{H}(x)$, is a simple example of a safe function for the class $\mathcal{C}_{\mathsf{time},T=t,q}$ of time bounded attackers – see Claim 20. The trade-off is sharp since it is trivial to compute $f(x)$ in sequential time $t+1$. This is a desirable property in our context since the encoder/decoder both need to compute $f(x)$ for a random input x.

We remark that the proof of Claim 20 is very similar to an argument of Cohen and Pietrzak [17]. Our bound is slightly tighter, but less general. Cohen and Pietrzak [17] proved that any pROM algorithm running in time t can produce an arbitrary H-sequence with probability at most $\mathcal{O}\left(\frac{q^2}{2^w}\right)$. We can reduce the bound to $\mathcal{O}\left(\frac{qt}{2^w}\right)$ since the attacker needs to compute a specific H-sequence i.e., $L_1, \ldots, L_{t+1}$ with $L_i = \mathsf{H}(x)^i$. In general, we may have $q \ll t$.

▷ Claim 20. Let $f(x) = \mathsf{H}(x)^{t+1}$ and let $\epsilon = (t+1)t/2^{w+1} + (qt+1)2^{-w}$ then the function f is ϵ-safe for the class $\mathcal{C}_{\mathsf{time},T=t,q}$.

Proof. (Sketch) Let $L_i := \mathsf{H}(x)^i$. We remark that if $L_1, \ldots, L_{j-1}$ are all distinct then

$$\mathbf{Pr}\left[L_j = \mathsf{H}(L_{j-1}) \in \{L_1, \ldots, L_{j-1}\right] \leq (j-1)2^{-w}$$

Thus, the probability of the event $\mathtt{COL}$ that $L_i = L_j$ for some $1 \leq i < j \leq t+1$ is at most $2^{-w}\sum_{j=1}^{t+1}(j-1) = (t+1)t/2^{w+1}$. We say that a particular random oracle query u in round i is lucky if the output is $\mathsf{H}(u) = L_j$ but the label L_{j-1} had not previously been observed as the output to any earlier random oracle query. If i denotes the maximum index such that L_i has been observed as a random oracle output, then the probability that a particular query u is lucky is at most $\Pr[\mathsf{H}(u) \in \{L_{i+2}, \ldots, L_{t+1}\} | \overline{\mathtt{COL}}] = (t-i)2^{-w} \leq t2^{-w}$.

Conditioning on the event $\overline{\mathtt{COL}}$ that no collisions occur, we can apply union bounds to show that, except with probability $qt/2^2$, there are no lucky queries. If there are no lucky queries, then after t sequential rounds the output $L_{t+1} = f(x)$ can be viewed as uniformly random and the probability that the attacker outputs $f(x)$ is at most 2^{-w} in this case. ◀

If we let r denote the maximum number of sequential calls to $\mathsf{H}(\cdot)$ that can be evaluated in a second[3] then we could set $t = r \times L_{max}$, where L_{max} denotes the maximum latency of the channel. Note that the encoder/decoder would need require time marginally higher than the latency $L_{max} + 1/r \approx L_{max}$ to compute $H^{t+1}(x)$.

4.3 Graph Labeling Functions

We define a labeling function $f_{G,H}(x)$ on a graph G, hash function H, and input x.

▶ **Definition 21.** *Given a DAG $G = (V = [N], E)$ and a random oracle function $H : \Sigma^* \to \Sigma^w$ over an alphabet Σ, we define the labeling of graph G as $L_{G,H} : \Sigma^* \to \Sigma^*$. In particular, given an input x the (H, x) labeling of G is defined recursively by*

$$L_{G,H,x}(v) = \begin{cases} H(v \circ x), & \mathsf{indeg}(v) = 0 \\ H\left(v \circ L_{G,H,x}(v_1) \circ \cdots \circ L_{G,H,x}(v_d)\right), & \mathsf{indeg}(v) > 0, \end{cases}$$

[3] Bonneau and Schechter [14] estimated that SHA256 can be evaluated $r \approx 10^7$ times per second on a single core processor

where $v_1, \ldots, v_d$ are the parents of v in G, according to some predetermined lexicographical order. We define $f_{G,H}(x) = \{L_{G,H,x}(s)\}_{s \in \mathsf{sinks}(G)}$. If there is a single sink node s_G then $f_{G,H}(x) = L_{G,H,x}(s_G)$. We omit the subscripts G, H, x when the dependency on the graph G and hash function H is clear.

The graph labeling function can be used to construct safe functions for several different classes of resource bounded adversaries. In particular, the resources necessary to compute $f_{G,H}$ in the pROM are tightly linked to the black pebbling cost of the DAG G.

Parallel Black Pebbling Game. A legal (parallel) pebbling $P = (P_0, P_1, \ldots, P_t)$ of a DAG $G = (V, E)$ consists of a sequence of pebbling configurations $P_i \subseteq V$ – representing the set of labels $L_{G,H,x}(v)$ which are stored in memory at time i. We start with no pebbles on the graph $P_0 = \emptyset$, and can remove pebbles from the graph (free memory) at any time. For any newly pebbled node $v \in P_{i+1} \setminus P_i$, it must be the case that $\mathsf{parents}(v) \subseteq P_i$ where $\mathsf{parents}(v) := \{u \ : \ (u, v) \in E\}$. Intuitively, this is because we cannot compute $L_{G,H,x}(v)$ unless each of the dependent values $L_{G,H,x}(u)$ for each $u \in \mathsf{parents}(v)$ is already available in memory. In the parallel version of the black pebbling game, there is no constraint on the number of new pebbles $\left|P_{i+1} \setminus P_i\right|$ that can be placed on the graph in each round.

The space cost of a pebbling P is defined as $\mathsf{space}(P) := \max_i |P_i|$ and the space complexity of a graph is $\mathsf{space}(G) = \min_P \mathsf{space}(P)$. The space-time (resp. cumulative cost) cost of a pebbling P is the product $\mathsf{space}-\mathsf{time}(P) := \mathsf{time}(P) \times \mathsf{space}(P)$ (resp. $\mathtt{CC}(G) = \sum_i |P_i|$). We remark that $\mathtt{CC}(G) \leq \mathsf{space}-\mathsf{time}(G)$. For constant degree graphs G with N nodes it is known that $\mathsf{space}(G) = \mathcal{O}\left(N/\log N\right)$ and that $\mathtt{CC}(G) = \mathcal{O}\left(N^2 \log\log N/\log N\right)$ [1]. One can also construct graphs G s.t. $\mathtt{CC}(G) = \Omega(N^2/\log N)$ [3, 2] and Paul et al. [34] constructed a constant indegree graph G with $\mathsf{space}(G) = \Omega(N/\log N)$ [34, 4] – this last bound is tight as Hopcroft et al. [26] showed that *any* static DAG G on N nodes with constant indegree can be pebbled with at most $\mathsf{space}(G) = \mathcal{O}\left(N/\log N\right)$ pebbles.

Pebbling Reductions. In the full version [13] we prove that if $\mathsf{space}(G) \geq m$ and $S = mw/2$ then $f_{G,H}$ is safe for the class $\mathcal{C}_{\mathsf{space},S,q}$. The pebbling reduction is conceptually very similar to the reduction of Alwen and Serbinenko [6] who proved that $\mathsf{CMC}(f_{G,H}) = \Omega(\mathtt{CC}(G) \cdot w)$ i.e., if the graph G has high cumulative pebbling cost then $f_{G,H}$ is safe for the class $\mathcal{C}_{\mathsf{CMC},M,q}$, and by extension safe for the class $\mathcal{C}_{\mathsf{space-time},M,q} \subseteq \mathcal{C}_{\mathsf{CMC},M,q}$. In particular, given an execution trace $\mathsf{Trace}_{\mathcal{A},R,\mathsf{H}}(x)$ for an algorithm $\mathcal{A}^{H(\cdot)}(x)$ computing $f_{G,H}(x)$ we can (with high probability) extract a legal pebbling $P = (P_1, \ldots, P_t)$ for G and then use an extractor argument to show that $|\sigma_i|/w \geq |P_i|/2$ during each round i – otherwise we could derive a contradiction by using the extractor to compress the random oracle. Thus, to construct a safe function one simply needs to find a graph G with sufficiently large pebbling cost.

4.4 Brief Note on Candidate Constructions without Random Oracles

Recall that the proof of correctness for our LDC constructions on space bounded channels uses the random oracle model inherently through an extractor argument showing that any space bounded channel that fools a decoding algorithm can also essentially predict a random string. Obtaining the same results without random oracles is an open challenge. We remark that there are several candidate constructions of safe-functions in the standard model e.g., using *time-lock puzzles* [36]. One may also be able to use the framework of Bitansky et al. [11] to construct safe-functions without random oracles e.g., Bitansky et al. construct explicit time-lock puzzles from the minimal assumption that "inherently sequential" languages exist. It is plausible that the same construction would also yield space-bound (or space-time bound) puzzles from minimal assumptions. See the full version [13] for additional assumptions.

References

1 Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 241–271, Santa Barbara, CA, USA, August 14–18 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_9.

2 Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1001–1017, Dallas, TX, USA, October 31 – November 2 2017. ACM Press. doi:10.1145/3133956.3134031.

3 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 3–32, Paris, France, April 30 – May 4 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-56617-7_1.

4 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 99–130, Tel Aviv, Israel, April 29 – May 3 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-78375-8_4.

5 Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 33–62, Paris, France, April 30 – May 4 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-56617-7_2.

6 Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 595–603, Portland, OR, USA, June 14–17 2015. ACM Press. doi:10.1145/2746539.2746622.

7 L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. Bpp has subexponential time simulations unless exptime has publishable proofs. In *[1991] Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 213–219, June 1991. doi:10.1109/SCT.1991.160263.

8 László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991. doi:10.1145/103418.103428.

9 Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 912–926, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

10 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. A preliminary version appeared in the Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC).

11 Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 345–356, Cambridge, MA, USA, January 14–16 2016. Association for Computing Machinery. doi:10.1145/2840728.2840745.

12 Jeremiah Blocki, Venkata Gandikota, Elena Grigorescu, and Samson Zhou. Relaxed locally correctable codes in computationally bounded channels. In *IEEE International Symposium on Information Theory, ISIT*, page (to appear), 2019.

13 Jeremiah Blocki, Shubhang Kulkarni, and Samson Zhou. On locally decodable codes in resource bounded channels. *CoRR*, abs/1909.11245, 2019.

14 Joseph Bonneau and Stuart E. Schechter. Towards reliable storage of 56-bit secrets in human memory. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014: 23rd USENIX Security Symposium*, pages 607–623, San Diego, CA, USA, August 20–22 2014. USENIX Association.

15 Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 402–414, 1999.

16 Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998. `doi:10.1145/293347.293350`.

17 Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 451–467, Tel Aviv, Israel, April 29 – May 3 2018. Springer, Heidelberg, Germany. `doi:10.1007/978-3-319-78375-8_15`.

18 A. Deshpande, R. Jain, T. Kavitha, S. V. Lokam, and J. Radhakrishnan. Better lower bounds for locally decodable codes. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 184–193, May 2002. `doi:10.1109/CCC.2002.1004354`.

19 Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM J. Comput.*, 40(4):1154–1178, 2011.

20 Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012.

21 Peter Gemmell, Richard Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 33–42, New York, NY, USA, 1991. ACM. `doi:10.1145/103418.103429`.

22 Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. In *9th Innovations in Theoretical Computer Science Conference, ITCS*, pages 27:1–27:11, 2018.

23 Venkatesan Guruswami and Adam Smith. Optimal rate code constructions for computationally simple channels. *J. ACM*, 63(4):35:1–35:37, September 2016. `doi:10.1145/2936015`.

24 Brett Hemenway and Rafail Ostrovsky. Public-key locally-decodable codes. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Proceedings*, pages 126–143, 2008.

25 Brett Hemenway, Rafail Ostrovsky, Martin J. Strauss, and Mary Wootters. Public key locally decodable codes with short keys. In *14th International Workshop, APPROX, and 15th International Workshop, RANDOM, Proceedings*, pages 605–615, 2011.

26 John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977. `doi:10.1145/322003.322015`.

27 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 80–86, New York, NY, USA, 2000. ACM. `doi:10.1145/335305.335315`.

28 Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *J. Comput. Syst. Sci.*, 69(3):395–420, 2004.

29 Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *J. ACM*, 64(2):11:1–11:42, 2017.

30 E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, October 1997. `doi:10.1109/SFCS.1997.646125`.

31 Richard J. Lipton. A new approach to information theory. In *STACS 94*, pages 699–708, Berlin, Heidelberg, 1994.

32 Silvio Micali, Chris Peikert, Madhu Sudan, and David A. Wilson. Optimal error correction against computationally bounded noise. In Joe Kilian, editor, *Theory of Cryptography*, pages 1–16, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

33 Rafail Ostrovsky, Omkant Pandey, and Amit Sahai. Private locally decodable codes. In *Automata, Languages and Programming*, pages 387–398, 2007.

34 Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 149–160, New York, NY, USA, 1976. ACM. doi:10.1145/800113.803643.

35 C. Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan 2009*, 2009.

36 Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, USA, 1996.

37 Ronen Shaltiel and Jad Silbak. Explicit list-decodable codes with optimal rate for computationally bounded channels. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 45:1–45:38, 2016.

38 Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. doi:10.1006/jcss.2000.1730.

39 Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008.