Weak Zero-Knowledge via the Goldreich-Levin Theorem

Dakshita Khurana[1], Giulio Malavolta[2,3], and Kabir Tomer[1]

[1] UIUC, Urbana-Champaign, USA
{dakshita, ktomer2}@illinois.edu
[2] Bocconi University, Milan, Italy
[3] Max Planck Institute for Security and Privacy, Bochum, Germany
giulio.malavolta@hotmail.it

Abstract. Obtaining three round zero-knowledge from standard cryptographic assumptions has remained a challenging open problem. Meanwhile, there has been exciting progress in realizing useful relaxations such as weak zero-knowledge, strong witness indistinguishability and witness hiding in two or three rounds. In particular, known realizations from generic assumptions obtain: (1) security against *adaptive* verifiers assuming fully homomorphic encryption among other standard assumptions (Bitansky et. al., STOC 2019), and (2) security against *non-adaptive* verifiers in the distributional setting from oblivious transfer (Jain et. al., Crypto 2017).
This work builds three round weak zero-knowledge for NP in the non-adaptive setting from doubly-enhanced injective trapdoor functions. We obtain this result by developing a new distinguisher-dependent simulation technique that makes crucial use of the Goldreich-Levin list decoding algorithm, and may be of independent interest.

Keywords: Distinguisher, Simulation, Zero-knowledge, Goldreich-Levin.

1 Introduction

Zero-knowledge (ZK) proofs are among the most widely used cryptographic primitives, with a rich history of study. In particular there has been significant recent interest in understanding the round complexity of zero-knowledge and its variants.

Zero-knowledge protocols are typically defined via the simulation paradigm. A simulator Sim is a polynomial-time algorithm that mimics the interaction of an adversarial verifier $\mathcal{V}^*$ with an honest prover $\mathcal{P}$. Sim has access to the verifier $\mathcal{V}^*$ and knows the statement, i.e., $x \in \mathcal{L}$, for an instance x of an NP language $\mathcal{L}$. Importantly, the simulator does not have access to any other "secret" information, including the (typically hard to find) NP witness for x.

The goal of the simulator is to generate a simulated view for the verifier $\mathcal{V}^*$, given only the information that $x \in \mathcal{L}$, such that this is indistinguishable from the view that the verifier would have obtained in its real interaction with the

honest prover $\mathcal{P}$. Informally, a protocol is zero-knowledge (ZK) iff there exists a simulator Sim that with access to $\mathcal{V}^*$ generates a view, which fools all distinguishers $\mathcal{D}^*$ that may be trying to distinguish a simulated view from a real one.

Weak zero-knowledge [10]. The definition of *weak zero-knowledge* (WZK) relaxes ZK by reversing the order of quantifiers: it requires that for every verifier-distinguisher pair $(\mathcal{V}^*, \mathcal{D}^*)$, there exists a simulator Sim that fools this pair. This removes the need for a universal simulator that fools all distinguishers. In applications, this reversal of quantifiers does not necessarily incur a large privacy cost: e.g., this still guarantees that no $(\mathcal{V}^*, \mathcal{D}^*)$ can recover the NP witness, or any predicate of the NP witness. In fact, it also implies the following relaxations.

- **Witness Hiding [12]** loosely guarantees that a malicious verifier cannot recover a witness from a proof unless the witness can be efficiently computed from the statement alone.
- **Strong Witness Indistinguishability (Strong WI) [13]** requires that for two indistinguishable statement distributions $\mathcal{D}_0, \mathcal{D}_1$, a proof (or argument) for statement $d_0 \leftarrow \mathcal{D}_0$ must be indistinguishable from a proof (or argument) for statement $d_1 \leftarrow \mathcal{D}_1$.
- **Witness indistinguishability (WI) [12]** ensures that proofs of the *same statement* generated using different witnesses are indistinguishable. WI does not hold for statements sampled from different distributions, and is meaningless for statements that have a unique witness associated with them.

Prior Techniques to Realize Weak ZK. Unlike zero-knowledge, weak ZK (and therefore all the other relaxations it implies) *has been shown to be achievable* in three rounds assuming the existence of *unleveled fully-homomorphic encryption* [4]. In a more relaxed "non-adaptive" setting, where the instance is sampled from an entropic distribution only *after the verifier's challenge is fixed*, three round weak ZK (and similarly, all other relaxations) can be obtained from weaker assumptions: namely any *statistically sender-private* (SSP) oblivious transfer. This type of OT can be instantiated from algebraic assumptions such as DDH [22,1], QR, N^{th} residuosity [18], LWE [5], and most recently even LPN [3]. Similarly, while the primary contribution of [4] is to remove the non-adaptivity requirement via fully homomorphic encryption, a pared-down version of their protocol yields weak ZK in the *non-adaptive* setting from *random self-reducible* public-key encryption (which can also be viewed as a type of homomorphism) as opposed to OT. Finally, a recent work [6] obtains a realization of distinguisher-dependent simulation under the *specific* assumption that factoring is hard.

At a high level, all the above works build strategies that *enable a simulator to learn a trapdoor* by making repeated queries to a distinguisher, called *distinguisher-dependent simulation*. This technique has subsequently had had applications to non-malleable cryptography [21] and MPC [2], to low-communication laconic protocols [7] and new types of oblivious transfer [20,8]. Despite its applicability, we lack an understanding of the *generic assumptions* under which this technique is instantiable.

This work: Non-adaptive Distinguisher-Dependent Simulation from Doubly-Enhanced Injective TDFs. This work focuses on improving our understanding of the generic assumptions that can be used to realize distinguisher-dependent simulation. In particular, while existing realizations from generic assumptions require schemes with homomorphic capabilities, we ask whether there are other classes of generic assumptions that imply three-round weak zero-knowledge protocols.

We obtain a positive answer to this question, obtaining non-adaptive weak ZK/strong witness indistinguishable/witness hiding arguments assuming the existence of enhanced, injective trapdoor functions and two-message witness indistinguishable arguments. The latter can themselves be based on doubly-enhanced injective trapdoor functions (with efficiently verifiable keys) [9] or an array of assumptions such as (subexponentially hard versions of) DDH, QR, N^{th} residuosity, LWE. Under the same assumptions, we also obtain three round *arguments of knowledge* that satisfy weak zero-knowledge, strong witness indistinguishability and witness hiding properties in the non-adaptive setting. Even in the non-adaptive setting, these systems are already known to have several applications, including to multiparty computation [19] and non-malleable commitments [21].

2 Our Techniques

The starting point of our work is the following template for distinguisher-dependent simulation, that abstracts out and generalizes ideas underlying existing frameworks.

A Template for Distinguisher-Dependent Simulation: Encrypted Proofs. Suppose a prover $\mathcal{P}$ wants to convince a verifier $\mathcal{V}$ that an instance $x \in \mathcal{L}$, for some NP language $\mathcal{L}$.

- The verifier $\mathcal{V}$ will first sample a puzzle together with a corresponding solution – which we will call the secret. $\mathcal{V}$ will send the puzzle to the prover $\mathcal{P}$, while keeping the secret hidden.
- Given a verifier message containing the puzzle, $\mathcal{P}$ will encrypt its proof in such a way that anyone that holds the secret corresponding to this puzzle can decrypt and check the proof.

For security, we will require that:

- Any proof encrypted to a puzzle cannot be decrypted without *knowledge* of the corresponding secret, and
- Given a random puzzle, its corresponding secret is hard to find.

The first requirement on *knowledge* can be made more precise: consider any distinguisher $\mathcal{D}^*$ that distinguishes between encrypted proofs generated by an honest prover, and proofs sampled (efficiently) by a simulator from a public distribution. Then it should be possible to *extract* a secret from this distinguisher

by building an efficient *search* algorithm $\mathcal{S}^*$. In more detail, $\mathcal{S}^*$ on input a random puzzle should be able to use $\mathcal{D}^*$ to efficiently find the secret for $\mathcal{V}^*$'s puzzle.

Finally, using known techniques [11], this protocol can be designed to allow a simulator to easily generate simulated views once it knows the secret. In summary, for any $\mathcal{D}^*$, either

- Proofs sampled by Sim from a public distribution already fool $\mathcal{D}^*$, and if not, then
- Sim can use the search algorithm $\mathcal{S}^*$ (which itself runs the verifier, and the distinguisher $\mathcal{D}^*$) to extract a secret that will enable Sim to fool $\mathcal{D}^*$.

What are some generic assumptions under which this template can be instantiated? To begin with, we observe that the template does appear necessitate (a form of) public-key encryption. Injective trapdoor functions are among the weakest generic primitives that are known to imply public-key encryption. In this work, we will aim to understand whether this template can be instantiated from trapdoored variants of injective one-way functions. Before studying this question, a few remarks about prior works are in order.

First, we note that in retrospect, prior works [19,4,6] can be viewed as instantiations of the generic template above under very specific homomorphic-style assumptions. For example, the protocol in [19] uses two-round oblivious transfer (OT) to ensure that a verifier obtains one out of two possible challenge openings for a $\Sigma-$protocol[4]. At a very high level, the simulation technique in [19] builds on the fact that (1) either simulated "garbage" $\Sigma-$protocol openings will already fool a distinguisher, or (2) if the distinguisher is able to meaningfully recover one (and only one, due to OT security) out of two $\Sigma-$protocol openings, then a simulator can find which of the two is being recovered by running such a distinguisher repeatedly, and this information can then be used to complete simulation. The work of [4] instead relies on random-self-reducible encryption towards a similar end: namely, (1) either a distinguisher cannot distinguish encryptions of 0 from encryptions of 1, or (2) if the distinguisher is able to distinguish between these encryptions, then it can be used to decrypt a specific challenge ciphertext, thereby helping the simulator find a trapdoor. As such, while the generic template discussed above itself does not appear to necessitate any homomorphic properties, these prior instantiations [19,4] require certain flavors of homomorphism (e.g., oblivious transfer is roughly equivalent to homomorphic encryption for certain linear functions) to reduce solving specific instances to solving random instances of a similar problem.

In contrast, in this work, we aim to add to the class of assumptions yielding distinguisher-dependent simulation, by relying new types of generic assumptions which do not a-priori satisfy self-reducibility or homomorphic properties. This requires us to rely on other mechanisms for search-to-decision: in particular, we develop new techniques that build on the Goldreich-Levin algorithm to enable distinguisher-dependent simulation.

[4] This basic protocol is repeated in parallel in [19] to reduce soundness error.

Towards Distinguisher-Dependent Simulation from Trapdoor One-way Functions. The Goldreich-Levin list decoding theorem provides a natural search to decision strategy for encryption schemes: it guarantees that any adversary that has better than negligible advantage in predicting the hardcore bit $\langle a, s \rangle$ for fixed secret a and random s, can be used to *find* a with overwhelming probability.

Our first idea is to encrypt proofs in the template discussed above, via Goldreich-Levin hardcore bits. Let us imagine that the prover (in an as-yet unspecified manner) generates an initial proof π attesting to the fact that x is in L. Each bit π_i of π will be encrypted by XOR-ing it with Goldreich-Levin hardcore bits, i.e., the prover will send for each i, the ciphertext

$$\mathsf{ct}_i = f(a_i), \pi_i \oplus \langle a_i, s_i \rangle, s_i$$

for $a_i, s_i \leftarrow \{0,1\}^\kappa$, and where f denotes an (injective) one-way function. Any $\mathcal{D}^*$ that distinguishes honestly generated encrypted proofs from encryptions of junk values implies a search algorithm $\mathcal{S}^*$ that inverts f . However, there is no way for an *honest* verifier to *decrypt* and check the proof π.

To remedy this, we set f to be a family of *trapdoored* functions, where the verifier samples the function family and the corresponding trapdoor. Namely, our protocol is modified so that the verifier samples

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$$

obtained by running the key generation algorithm for a trapdoor (injective) one-way function family, and sends pk to the prover. Next, the prover computes

$$\mathsf{ct}_i = f_{\mathsf{pk}}(a_i), \pi_i \oplus \langle a_i, s_i \rangle, s_i$$

and the verifier decrypts these ciphertexts to obtain π given the corresponding secret key sk.

Now, given any $\mathcal{D}^*$ that distinguishes ciphertexts that encrypt well-formed proofs from ciphertexts that encrypt 0, it becomes possible to extract *inverses* $a_1, \ldots a_n$ of the one-way function outputs $f(a_1), \ldots f(a_n)$ contained in the ciphertexts.

We can label these values $(a_1, \ldots a_n)$ as the secret, and use these to provide an alternative path to simulation. Namely, we will modify the proof itself (also simplifying it along the way to use only one a value) so that the verifier sends the prover a public key pk as before. Next, the prover computes

$$\mathsf{ct} = (y_1, y_2) \text{ for } y_1 = f_{\mathsf{pk}}(a), y_2 = 0 \oplus \langle a, s \rangle, s$$

as an encryption of 0, and additionally computes a witness indistinguishable (WI) proof attesting to the fact that

"either $x \in L$, or ct *encrypts* 1*, or the prover knows an inverse $'a'$ of y_1"*

The verifier accepts this proof if the WI proof accepts, and also ct decrypts to 0 (the verifier can decrypt ct since it knows the trapdoor for the one-way function).

Fixing Soundness via Coin-Tossing and Enhanced TDFs. To ensure that the prover cannot get away with generating accepting proofs for $x \notin L$, we must ensure that the prover actually *does not* know the inverse a of y_1. This is not true for the protocol described so far – in particular, the prover can always first sample a, then sample y_1 as $f_{\mathsf{pk}}(a)$. To prevent this, we modify the protocol to perform a *coin-toss* between the prover and verifier, namely, $\mathcal{P}$ first commits to randomness r_0, then $\mathcal{V}$ sends randomness r_1, and finally $\mathcal{P}$ outputs a WI proof attesting to the fact that

"either $x \in L$, or ct *encrypts* 1*, or the prover knows an inverse $'a'$ of $r_0 \oplus r_1$"*

With this modification, it becomes possible to show that $r_0 \oplus r_1$ appears sufficiently random to the prover, and the prover is unable to find an inverse of $r_0 \oplus r_1$. However, to allow simulation to go through, we still need such an inverse to *exist* for most choices of $r_0 \oplus r_1$, which may not be true if we simply treat $r_0 \oplus r_1$ as an element of the co-domain.

As such, instead of requiring the prover to find an inverse a of $r_0 \oplus r_1$, we will have the prover use randomness $r = r_0 \oplus r_1$ to sample an element y_1 from the *image of the trapdoor function*, and the WI proof will ask to find an inverse a of y_1. That is, the prover (as before) computes

$$\mathsf{ct} = (y_1, y_2) \text{ for } y_1 = f_{\mathsf{pk}}(a), y_2 = 0 \oplus \langle a, s \rangle, s$$

as an encryption of 0, and computes a witness indistinguishable (WI) proof attesting to the updated statement

"either $x \in L$, or ct *encrypts* 1*,*
or the prover knows an inverse $'a'$ of y_1 for y_1 sampled from the image using randomness $r_0 \oplus r_1$"

As before, the verifier accepts this proof if the WI accepts, and also ct decrypts to 0 (the verifier can decrypt ct since it has a trapdoor for the one-way function).

With this modification, we are able to prove soundness assuming that it is hard to invert such a y_1, even given the randomness r used to sample y_1 (a trapdoor function satisfying this property is called an *enhanced* trapdoor function [16]).

Building a Distinguisher-Dependent Simulator. A simulator Sim given a statement x (and without knowing a witness w for $x \in L$), has the following options:

- Generate ct as an encryption of 1, and use this as witness for the WI proof. If $\mathcal{D}^*$ cannot distinguish such a ciphertext from an honestly generated ciphertext (encrypting 0), then this option succeeds and the simulator's job is done.
- On the other hand, suppose the first option fails because $\mathcal{D}^*$ distinguishes a ciphertext encrypting 1 from the honestly generated one (encrypting 0). Then the Goldreich-Levin theorem suggests that $\mathcal{D}^*$ can be used by Sim

to compute inverses. In particular Sim can generate $\mathsf{ct} = (y'_1, y'_2)$ for $y'_1 = r_0 \oplus r_1$, and then use $\mathcal{D}^*$ to find its inverse a. This also requires Sim to sample other instance-witness pairs from the distribution on its own, which is why our techniques are limited to the non-adaptive setting. Finally, Sim can use a extracted above as a witness for the WI proof, thereby successfully completing simulation.

A Technical Subtlety. We discuss one additional technical subtlety that requires us to further modify the protocol sketched above. First, the definition of enhanced trapdoor function families guarantees that *honestly sampled keys* $(\mathsf{pk}, \mathsf{sk})$ lead to invertible distributions on y. But a malicious verifier may sample pk for which y values sampled as above *do not* have an inverse. This would cause the simulation strategy described above to break down.

We address this issue by relying on two-round witness indistinguishable (WI) arguments. In particular, the protocol above is modified to have the verifier generate two sets of keys $(\mathsf{pk}_1, \mathsf{sk}_1)$ and $(\mathsf{pk}_2, \mathsf{sk}_2)$ and prove (via a WI argument) that one of these pairs is well-formed, and will necessarily lead to invertible samples. We show that the resulting protocol is both simulatable and an argument of knowledge by combining all the techniques discussed above with the two-key technique [23].

This completes an overview of our protocol, where we assumed the existence of (1) enhanced trapdoor functions and (2) two-round witness indistinguishable arguments. For the sake of brevity, we swept some additional technical details under the rug; we point the reader to Section 4 for a more detailed description of our protocol. We conclude this section with a couple of natural problems that would be useful to address in future research.

Open Problems and Directions for Future Work. A natural open problem given our work and prior works, is to understand whether distinguisher-dependent simulation can be realized based on any public-key encryption scheme. Second, in light of the generic template discussed earlier in the overview, it is reasonable to wonder whether distinguisher-dependent simulation can be realized in minicrypt, or if public-key encryption is *necessary*. We conjecture that two-round proofs with distinguisher-dependent simulation imply public-key cryptography, but leave a formal exploration of this question for future work. As these questions demonstrate, there remain gaps in our understanding of what assumptions are necessary and sufficient for (non-trivial) distinguisher-dependent protocols in two or three rounds. Nevertheless, we believe that demonstrating the utility of the Goldreich-Levin search-to-decision reduction as is done in this work may be a useful step towards answering some of these questions.

3 Preliminaries

Notation. Given an NP relation R, we denote the NP language associated with it as $L_R := \{x : \exists w \text{ such that } R(x, w) = 1\}$.

3.1 Goldreich-Levin List Decoding

In what follows, we recall the Goldreich-Levin Theorem [15], borrowing some text verbatim from [17].

Definition 1 (Goldreich-Levin Bit Prediction Algorithm [15]). *Fix $\varepsilon > 0$ and a secret $x \in \{0,1\}^\kappa$. A probabilistic Goldreich-Levin prediction algorithm* $\mathsf{GL.Pred}$ *for secret x with advantage ε takes input $r \in \{0,1\}^\kappa$ and outputs a value in $\{0,1\}$ such that:*

$$\Pr_{r \xleftarrow{R} \{0,1\}^\kappa} [\mathsf{GL.Pred}(r) = \langle x, r \rangle] \geq 1/2 + \varepsilon$$

where probability is taken over r and the randomness of $\mathsf{GL.Pred}$.

Theorem 1 (Goldreich-Levin Theorem [15]). *For any given $\varepsilon > 0$ and any polynomial-time computable function $f : \{0,1\}^\kappa \to \{0,1\}^{poly(\kappa)}$, there exists an algorithm* $\mathsf{GL.Inv}$ *that runs in time $poly(1/\varepsilon, \kappa)$ with the following property.*

Let $\mathsf{GL.Pred}$ *be a Goldreich-Levin prediction algorithm for secret x with advantage ε (Definition 1). Then* $\mathsf{GL.Inv}$*, given oracle access to* $\mathsf{GL.Pred}$*, queries the oracle at most $poly(1/\varepsilon, \kappa)$ times and outputs x with probability $1 - 2^{-\Omega(\kappa)}$. That is,*

$$\Pr[\mathsf{GL.Inv}^{\mathsf{GL.Pred}}(f(x)) = x] \geq 1 - \mathsf{negl}(\kappa)$$

where probability is taken over the randomness of $\mathsf{GL.Inv}$.

3.2 Building Blocks

Definition 2 (Enhanced trapdoor injective one-way functions [14]). *A family of enhanced trapdoor injective one-way functions is a collection of injective functions $f_\alpha : \{D_\alpha \to R_\alpha\}$ such that:*

- **Syntax.** *There exist randomized PPT algorithms:*
  - *I such that $I(1^n) \to (\alpha, \tau)$ where α is the index of the injective function f_α, and τ is a corresponding trapdoor for the function.*
    We will denote by $I_1(\cdot)$ the function that runs $I(\cdot)$ and only outputs the first coordinate α, and by $I_2(\cdot)$ the function that runs $I(\cdot)$ and only outputs the second coordinate τ.
  - *S_D that on input α outputs an element from the domain D_α of the function f_α.*
  - *S_R that on input α outputs an element from the range R_α of f_α.*
- **Enhancement.** *For every $\alpha \in \mathsf{Supp}(I_1)$, the distributions $S_R(\alpha)$ and $f_\alpha(S_D(\alpha))$ are computationally indistinguishable.*
- **Efficiency.** *There exists a deterministic polynomial-time algorithm F such that for all $\alpha \leftarrow I_1(1^\kappa)$, and for all $x \in D_\alpha$, $F(\alpha, x) = f_\alpha(x)$.*
- **Trapdoor Inversion.** *There exists a deterministic polynomial-time algorithm B such that for all $(\alpha, \tau) \leftarrow I(1^\kappa)$, and for all $y \in R_\alpha$, $B(\tau, y)$ outputs an x such that $f_\alpha(x) = y$.*

– **Security (Hardness of Inversion).** *This guarantees that samples output by the* range sampler S_R *are hard to invert, even given the randomness used to sample them.*

$$\Pr\left[f_\alpha(x') = y \;\middle|\; \begin{array}{c} \alpha \leftarrow I_1(1^n) \\ r \xleftarrow{R} \{0,1\}^* \\ y \leftarrow S_R(\alpha; r) \\ x' \leftarrow A(\alpha, r) \end{array} \right] \leq \mathsf{negl}(n)$$

We note that these types of definitions have previously naturally arisen in the study of oblivious transfer and non-interactive zero-knowledge, where sometimes a strengthening to *permutations* is considered. It is shown in [16] that natural versions of the RSA and Rabin collections of trapdoor functions satisfy the definition above, and in fact also yield doubly enhanced trapdoor permutations. In this work, we will not require permutations, and can work with the weaker definition above (this weakening has also been considered previously, e.g., in [16]).

Definition 3 (Perfectly Binding Non-Interactive Bit-Commitments). *A perfectly binding non-interactive bit-commitment scheme consists of a PPT algorithm* com *such that:*

– **Perfect Binding:** *For all* $\kappa \in \mathbb{N}$, $\forall r_0, r_1 \in \{0,1\}^\kappa$, *and* $\forall b_0, b_1 \in \{0,1\}$*:*

$$(\mathsf{com}(b_0; r_0) = \mathsf{com}(b_1; r_1)) \implies (b_0 = b_1)$$

– **Computational Hiding:** *For all non-uniform PPT adversaries* A*, there exists a negligible function* μ *such that for all* $\kappa \in \mathbb{N}$*:*

$$\Pr_{r \xleftarrow{R} \{0,1\}^\kappa, b \xleftarrow{R} \{0,1\}} [b' = b | b' \leftarrow A(1^\kappa, \mathsf{com}(b; r))] \leq \frac{1}{2} + \mu(\kappa)$$

Bit commitments can be used to commit to strings of length $\mathsf{poly}(\kappa)$ by separately committing to each bit of the string. This preserves perfect binding and computational hiding.

Furthermore, perfectly binding non-interactive bit-commitments can be based on any injective one-way trapdoor function family, simply by masking the input with the Goldreich-Levin hardcore bit of the function.

3.3 Proof Systems

In this section, we recall definitions of proof systems, including delayed-input protocols and weak-zero knowledge following [19].

Definition 4 (Delayed-Input Interactive Protocols [19]). *An* n*-round delayed-input interactive protocol* (P, V) *for deciding a language* L *associated with the relation* R_L *is an interactive protocol for the same where:*

– *To prove $x \in L$, the prover and the verifier initially receive the size of the instance and execute the first $n-1$ rounds of the protocol.*
– *At the start of the last round, the prover receives $(x, w) \in R_L$ and the verifier receives x. They then execute the last round of the protocol.*

Definition 5 (Delayed-Input Interactive Arguments [19]). *An n-round delayed-input interactive argument (P, V) for deciding a language L associated with the relation R_L is an interactive protocol such that it satisfies the following:*

– **Completeness:** *For all $(x, w) \in R_L$,*

$$\Pr[\langle P, V\rangle(x, w) = 1] \geq 1 - \mathsf{negl}(\kappa)$$

where probability is taken over the randomness of P and V.
– **Adaptive Soundness:** *For every κ, for every PPT P^* that chooses an $x \in \{0,1\}^\kappa \setminus L$ adaptively after the first $n-1$ rounds of the protocol,*

$$\Pr[\langle P^*, V\rangle(x) = 1] \leq \mathsf{negl}(\kappa)$$

where probability is taken over the randomness of P^* and V.

Definition 6 (Witness Indistinguishability [19]). *An n-round delayed-input interactive argument (P, V) for deciding a language L associated with the relation R_L is said to be witness-indistinguishable if for every non-uniform PPT verifier V^* and all (x, w_1, w_2) where $w_1, w_2 \in R_L(x)$, the following ensembles are computationally indistinguishable:*

$$\{\langle P, V^*\rangle(x, w_1)\} \text{ and } \{\langle P, V^*\rangle(x, w_2)\}$$

Two-round witness indistinguishable arguments can be based on the existence of doubly-enhanced injective trapdoor functions [9,16].

Definition 7 (Argument of Knowledge). *A delayed-input interactive argument is an argument of knowledge if there exists a polynomial time extractor E such that for any polynomial-size prover P^*, there exists a negligible μ such that for any security parameter $\kappa \in \mathbb{N}$,*

$$\Pr\left[\begin{matrix} V(x;\tau) = 1 \wedge \\ w \notin R_L(x) \end{matrix} \middle| \begin{matrix} (x,\tau) \leftarrow \langle P^*, V\rangle \\ w \leftarrow E^{P^*}(x,\tau) \end{matrix}\right] \leq \mu(\kappa).$$

A *witness indistinguishable argument of knowledge* is a proof system that satisfies both the witness indistinguishability and the argument of knowledge properties above. We now define what it means for an argument of knowledge to satisfy reusable witness indistinguishability.

Definition 8 (Reusable Witness Indistinguishable Argument of Knowledge [19]). *An n-round delayed input interactive argument of knowledge (P, V) for a language L is Reusable Witness Indistinguishable if for all PPT V^*, all $k = \mathsf{poly}(\kappa)$, $\Pr[b = b'] \leq 1/2 + \mathsf{negl}(\kappa)$ for the following game.*

- (P, V^*) *initially receive the size of the instance and execute the first* $n-1$ *rounds.*
- V^* *then outputs* $(x^1, w^1), (x^2, w^2), ..., (x^{k-1}, w^{k-1})$.
- P *generates the* n^{th} *message of the delayed-input witness indistinguishable argument of knowledge for the instances* $(x^1, x^2, ..., x^{k-1})$ *using the witnesses* $(w^1, w^2, ...w^{k-1})$ *and sends them to* V^*.
- V^* *outputs* (x^k, w_1^k, w_2^k).
- P *samples a single bit* b *and generates the* n^{th} *message of the delayed-input WIAoK for the instance* x^k *using witness* w_b^k.
- V^* *outputs* b'.

Reusable WI arguments of knowledge can be based on the existence of two-round witness indistinguishable arguments (ZAPs), as shown in [19].

Definition 9 (Non-adaptive Distributional ϵ-Weak Zero Knowledge [19]). *A delayed-input interactive argument* (P, V) *for a language* L *is said to be* distributional ϵ-weak zero knowledge against non-adaptive verifiers *if there exists a simulator* $\mathcal{S}$*, which is an oracle-aided machine that runs in time* $\mathsf{poly}(\kappa, \epsilon)$ *such that for every efficiently sampleable distribution* $(\mathcal{X}_\kappa, \mathcal{W}_\kappa)$ *on* R_L*, i.e.,* $\mathsf{Supp}(\mathcal{X}_\kappa, \mathcal{W}_\kappa) = \{(x, w) : x \in L \cap \{0,1\}^\kappa, w \in R_L(x)\}$*, every non-adaptive polynomial-size verifier* V^**, every polynomial-size distinguisher* $\mathcal{D}$*, and every* $\epsilon = 1/\mathsf{poly}(\kappa)$*,*

$$\left| \Pr_{(x,w)\leftarrow(\mathcal{X}_\kappa,\mathcal{W}_\kappa)}\left[\mathcal{D}(x, \mathsf{View}_{V^*}[\langle P, V^*\rangle(x,w)]) = 1\right] - \Pr_{(x,w)\leftarrow(\mathcal{X}_\kappa,\mathcal{W}_\kappa)}\left[\mathcal{D}(x, \mathcal{S}^{V^*,D}(x)) = 1\right] \right| \leq \epsilon(\kappa),$$

where the probability is over the random choices of (x, w) *as well as the random coins of the parties.*

4 Construction

In this section, we describe our protocol. We will use the following ingredients.

- Let f be a family of enhanced one-way trapdoor functions according to Definition 2, and denote the index sampler, domain and range samplers, function description and inversion function respectively by $(I = (I_1, I_2), S_D, S_R, F, B)$.
- Let com be a non-interactive, statistically binding, computationally hiding commitment scheme (Definition 3).
- Let $\mathsf{ZAP} = (\mathsf{ZAP}_1, \mathsf{ZAP}_2, \mathsf{ZAP}_{\mathsf{verify}})$ denote the verifier and prover next-message functions, and the verification algorithm respectively for a two-round witness indistinguishable argument (Definition 6) for NP. This will be used to prove membership in the language L_{zap} defined below.

– Let $\mathsf{WI} = (\mathsf{WI}_1, \mathsf{WI}_2, \mathsf{WI}_3, \mathsf{WI}_{\mathsf{verify}})$ denote the prover, verifier, and prover next-message functions, and the verification algorithm respectively for a 3-round delayed-input reusable witness indistinguishable argument of knowledge (Definition 8) for NP. This will be used to prove membership in the language L_{wi} defined below.

We fix the following predicates for notational convenience.

– Intuitively, the first predicate ϕ_{chal} takes in two tuples $(y_d, \mathsf{ik}_d, \mathsf{chal}_d)$ for $d \in \{0,1\}$, and ensures that for each d, chal_d "encrypts" b_d where $b_0 \neq b_1$, i.e. chal_d is of the form $F(\mathsf{ik}_d, y_d), b_d \oplus \langle y_d, r_d \rangle, r_d$. That is,

$$\begin{aligned}&\phi_{\mathsf{chal}}(y_0, y_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{chal}_0, \mathsf{chal}_1) = 1\\ &\iff \exists \widetilde{r}_0, \widetilde{r}_1, b_0, b_1 \text{ s.t. } \begin{pmatrix} \mathsf{chal}_0 = [F(\mathsf{ik}_0, y_0), b_0, \widetilde{r}_0] \ \wedge \\ \mathsf{chal}_1 = [F(\mathsf{ik}_1, y_1), b_1, \widetilde{r}_1] \ \wedge \\ b_0 \oplus b_1 = \langle y_0, \widetilde{r}_0 \rangle \oplus \langle y_1, \widetilde{r}_1 \rangle \oplus 1 \end{pmatrix}\end{aligned}$$

– The second predicate ϕ_{inv} on input a commitment c and strings r_0, r_1, s checks that c is a commitment to r_0 with randomness s; and that its remaining input z is an inverse of $r_0 \oplus r_1$ with respect to atleast one of its two input index keys $\mathsf{ik}_0, \mathsf{ik}_1$. Formally,

$$\begin{aligned}&\phi_{\mathsf{inv}}(r_0, s, z, r_1, \mathsf{ik}_0, \mathsf{ik}_1, c) = 1\\ &\iff \ (c = \mathsf{com}(r_0; s)) \wedge \begin{pmatrix} F(\mathsf{ik}_0, z) = S_R(\mathsf{ik}_0; r_0 \oplus r_1) \ \vee \\ F(\mathsf{ik}_1, z) = S_R(\mathsf{ik}_1; r_0 \oplus r_1) \end{pmatrix}\end{aligned}$$

We also define the following two languages.

– Let L_{zap} be a language with corresponding relation $R_{L_{\mathsf{zap}}}$ defined as:

$$R_{L_{\mathsf{zap}}}(x_{\mathsf{zap}}, r_{\mathsf{zap}}) = 1 \iff (x_0 = I_1(r_{\mathsf{zap}})) \vee (x_1 = I_1(r_{\mathsf{zap}}))$$

where x_{zap} is parsed as (x_0, x_1).

– Let L_{wi} be a language with corresponding relation $R_{L_{\mathsf{wi}}}$ defined as:

$$R_{L_{\mathsf{wi}}}(x_{\mathsf{wi}}, w_{\mathsf{wi}}) = 1 \iff \begin{pmatrix} (x, w) \in R_L \ \vee \\ \phi_{\mathsf{chal}}(y_0, y_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{chal}_0, \mathsf{chal}_1) \ \vee \\ \phi_{\mathsf{inv}}(r_0, s, z, r_1, \mathsf{ik}_0, \mathsf{ik}_1, c) \end{pmatrix}$$

where L is a given language, x_{wi} is parsed as $(x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and w_{wi} is parsed as (w, y_0, y_1, r_0, s, z).

5 Proof of Security

In this section, we prove the following theorem.

Theorem 2. *Assuming the existence of enhanced injective trapdoor functions satisfying Definition 2 and two-round witness indistinguishable arguments satisfying Definition 6, the protocol in Figure 1 satisfies non-adaptive distributional ϵ-weak zero knowledge.*

Weak Zero-Knowledge

Prover Input: An instance $x \in L$ and witness w s.t. $R_L(x, w) = 1$
Verifier Input: Language L

- **Prover Message:**
  1. Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa, s \xleftarrow{R} \{0,1\}^\kappa, r_{\mathsf{wi}} \leftarrow \{0,1\}^\kappa$.
  2. Compute $c := \mathsf{com}(r_0; s), \mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa), \mathsf{wi}_1 \leftarrow \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$.
  3. Send message $m_1 := (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to the verifier.
- **Verifier Message:**
  1. Sample $r_1 \xleftarrow{R} \{0,1\}^\kappa$
  2. Sample $(\mathsf{ik}_0, \tau_0) \xleftarrow{R} I(1^\kappa; r_{\mathsf{ik}_0})$ and $(\mathsf{ik}_1, \tau_1) \xleftarrow{R} I(1^\kappa; r_{\mathsf{ik}_1})$ where r_{ik_0} and r_{ik_1} are uniformly sampled strings.
  3. Set $x_{\mathsf{zap}} := (\mathsf{ik}_0, \mathsf{ik}_1)$ and $w_{\mathsf{zap}} := r_{\mathsf{ik}_0}$
  4. Compute $\mathsf{wi}_2 \leftarrow \mathsf{WI}_2(\mathsf{wi}_1)$ and $\mathsf{zap}_2 \leftarrow \mathsf{ZAP}_2(\mathsf{zap}_1, x_{\mathsf{zap}}, w_{\mathsf{zap}})$ proving that $R_{L_{\mathsf{zap}}}(x_{\mathsf{zap}}, w_{\mathsf{zap}}) = 1$
  5. Send message $m_2 := (r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$ to the prover.
- **Prover Message:**
  1. If $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ rejects, abort execution.
  2. Sample $q_0 \xleftarrow{R} S_D(\mathsf{ik}_0)$, $q_1 \xleftarrow{R} S_D(\mathsf{ik}_1)$, $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$
  3. Compute $\mathsf{chal}_0 = [F(\mathsf{ik}_0, q_0), \langle q_0, \widetilde{r}_0 \rangle, \widetilde{r}_0]$, $\mathsf{chal}_1 = [F(\mathsf{ik}_1, q_1), \langle q_1, \widetilde{r}_1 \rangle, \widetilde{r}_1]$
  4. Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := w$.
  5. Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$ proving that $R_{L_{\mathsf{wi}}}(x_{\mathsf{wi}}, w_{\mathsf{wi}}) = 1$.
  6. Send $m_3 := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$ to the verifier.
- **Verifier Output:** The verifier uses the inversion trapdoors τ_0, τ_1 to perform the following checks, and outputs 1 iff
  - There exist y_0, y_1 s.t. $\mathsf{chal}_0 = [F(\mathsf{ik}_0, y_0), \langle y_0, \widetilde{r}_0 \rangle, \widetilde{r}_0]$, $\mathsf{chal}_1 = [F(\mathsf{ik}_1, y_1), \langle y_1, \widetilde{r}_1 \rangle, \widetilde{r}_1]$, and
  - $\mathsf{WI}_{\mathsf{verify}}(\mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3)$ accepts.

Fig. 1: A Non-adaptive Distributional ϵ-weak Zero Knowledge Argument

5.1 Simulator

For language L with corresponding NP relation R_L, an efficiently sampleable distribution $(\mathcal{X}_\kappa, \mathcal{W}_\kappa)$ on R_L, an adversarial verifier A, a distinguisher $\mathcal{D}$, and an $\epsilon = 1/\mathsf{poly}(\kappa)$, we describe the following $\mathsf{poly}(\kappa, \epsilon)$ time simulator S, where $\epsilon' := \epsilon/7$. The simulator is described in stages below.

Initialization:
In this section the simulator behaviour is identical to an honest prover.

- Receive x as input.
- Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
- Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$

Algorithm $\mathsf{Pred}_{\gamma,\mathcal{Q},\mathsf{b}}(r)$

The algorithm $\mathsf{Pred}_{\gamma,\mathcal{Q},\mathsf{b}}$ takes an input r and is parameterised by the inversion target γ, the state $\mathcal{Q}$ and the bit b that chooses an index key.

$\mathcal{Q}$ is parsed as $(r_{\mathsf{wi}}, m_1, m_2, Q)$ where m_1 is parsed as $(c, \mathsf{zap}_1, \mathsf{wi}_1)$ and m_2 as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.

Pred does the following:

- Initialize A to state Q.
- Sample $\widetilde{b} \xleftarrow{R} \{0,1\}$
- Sample $q \xleftarrow{R} S_D(\mathsf{ik}_{1-\mathsf{b}})$.
- Sample $r' \xleftarrow{R} \{0,1\}^\kappa$
- Sample $(x', w') \xleftarrow{R} (\mathcal{X}, \mathcal{W})$
- Compute $\mathsf{chal}_\mathsf{b} = \left[\gamma, \widetilde{b}, r\right], \mathsf{chal}_{1-\mathsf{b}} = \left[f_{\mathsf{ik}_{1-\mathsf{b}}}(q), \langle q, r'\rangle, r'\right]$
- Set $x'_{\mathsf{wi}} := (x', \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w'_{\mathsf{wi}} := (w', \bot, \bot, \bot, \bot, \bot)$.
- Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x'_{\mathsf{wi}}, w'_{\mathsf{wi}}, L_{\mathsf{wi}})$.
- Send $x', \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3$ to the verifier.

If the output of $\mathcal{D}$ on input the view of the verifier is 1, output $\widetilde{b}$, else output $1 - \widetilde{b}$

Fig. 2: Predicting the hardcore bit for γ w.r.t. ik_b. Intuitively, m_1 and m_2 represent the first two messages of an execution of the protocol from Figure 1, and Q is the state of the verifier $\mathcal{A}$ from the protocol just before $\mathcal{A}$ receives the third message, after having received m_1 and sent m_2. r_{wi} is interpreted as the randomness used to generate the first WI message wi_1. Pred uses the distinguisher $\mathcal{D}$ to guess the Goldreich-Levin hardcore bit for challenge γ for input r, w.r.t. ik_b.

- Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary A.
- Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
- Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output verifier view if verification fails.

Let Q be the state of the adversary after the above. Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$.

Key Check Stage:
The simulator will perform a test on both index keys in order to choose one to use for inversion. The test aims to choose a key ik *such that* $\mathcal{D}$ *does not distinguish between a challenge ciphertext generated by calling* f_{ik} *on the output of* $S_D(\mathsf{ik})$ *or one generated by* $S_R(\mathsf{ik})$.

- Compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_\mathcal{Q}$ (from Figure 4) , where $b \in \{0, 1, \bot\}$
- If $b = \bot$, return $\bot$
- Else, continue onto the next stage.

Algorithm $\mathsf{Inv}(\gamma, \mathcal{Q}, \mathsf{b}, \varepsilon)$

The algorithm Inv takes as input r, inversion target γ, state $\mathcal{Q}$, a bit b that chooses an index key, and prediction advantage ε.

$\mathcal{Q}$ is parsed as $(r_{\mathsf{wi}}, m_1, m_2, Q)$ where m_1 is parsed as $(c, \mathsf{zap}_1, \mathsf{wi}_1)$ and m_2 as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.

Inv runs $\mathsf{GL.Inv}$ from Theorem 1 with oracle access to $\mathsf{Pred}_{\gamma,\mathcal{Q},\mathsf{b}}$. More formally, Inv performs the following steps:

- For advantage ε and the polytime-computable function $f_{\mathsf{ik_b}} : D_{\mathsf{ik_b}} \to R_{\mathsf{ik_b}}$, let $\mathsf{GL.Inv}$ be the Goldreich-Levin inversion algorithm given by Theorem 1.
- Construct an oracle P that answers queries by executing algorithm $\mathsf{Pred}_{\gamma,\mathcal{Q},\mathsf{b}}$.
- Compute $x = \mathsf{GL.Inv}^P(\gamma)$
- If $f_{\mathsf{ik_b}}(x) = \gamma$ return x, else return $\perp$.

Fig. 3: Inverting the trapdoor one-way function using Pred and Goldreich-Levin.

Inversion Stage:
The simulator will try to use the adversarial verifier and distinguisher to compute q' such that $f_{\mathsf{ik}_b}(q') = (S_R(\mathsf{ik}_b; r_0 \oplus r_1))$, where b is the output of $\mathsf{ik\text{-}check}$ *in the previous stage.*

- Run $\mathsf{Inv}(S_R(\mathsf{ik}_b; r_0 \oplus r_1), \mathcal{Q}, b, \epsilon'/2)$ (from Figure 3).
- If Inv succeeds and outputs some q' and proceed to the *Success Stage*.
- If Inv fails, proceed to *Failure Stage*.

Failure Stage:
If Inv *fails to invert during the* Inversion Stage*, then intuitively the distinguisher output does not depend on the Goldreich-Levin hardcore bit in the challenge with sufficient probability. This can be used to produce a transcript using the second branch of* WI*, by modifying the challenge. This requires knowledge of the inverse of the challenge. Since the inverse is not known for the previous challenge, a new challenge that fails inversion must be found.*

For each $i \in [\kappa/\epsilon']$:

- Rewind the adversary to state Q (unless already in state Q).
- Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
- If Inv fails:
  - Rewind the adversary to state Q.
  - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
  - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle \oplus 1, \widetilde{r}_b]$.

Algorithm $\mathsf{ik\text{-}check}_{\mathcal{Q}}$

The algorithm $\mathsf{ik\text{-}check}_{\mathcal{Q}}$ is parameterized by a state $\mathcal{Q}$ parsed as $(r_{\mathsf{wi}}, m_1, m_2, Q)$ where m_1 is parsed as $(c, \mathsf{zap}_1, \mathsf{wi}_1)$ and m_2 as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.

- Initial A in state Q.
- Let $n' := (4/\epsilon')^3$ and $t := n'\epsilon'/4$
- Initialize $c_{0,D} \leftarrow 0, c_{0,R} \leftarrow 0, c_{1,D} \leftarrow 0, c_{1,R} \leftarrow 0$
- For $i \in [n']$:
  - Sample $q_0 \xleftarrow{R} S_D(\mathsf{ik}_0), q_1 \xleftarrow{R} S_D(\mathsf{ik}_1)$
  - Sample $\gamma_0 \xleftarrow{R} S_R(\mathsf{ik}_0), \gamma_1 \xleftarrow{R} S_R(\mathsf{ik}_1)$
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_0}(q_0), \mathcal{Q}, 0, \epsilon'/2)$ and add 1 to $c_{0,D}$ if it succeeds.
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_1}(q_1), \mathcal{Q}, 1, \epsilon'/2)$ and add 1 to $c_{1,D}$ if it succeeds.
  - Run $\mathsf{Inv}(\gamma_0, \mathcal{Q}, 0, \epsilon'/2)$ and add 1 to $c_{0,R}$ if it succeeds.
  - Run $\mathsf{Inv}(\gamma_1, \mathcal{Q}, 1, \epsilon'/2)$ and add 1 to $c_{1,R}$ if it succeeds.
- Compute $\Delta_0 := |c_{0,D} - c_{0,R}|$
- Compute $\Delta_1 := |c_{1,D} - c_{1,R}|$

We say ik_b passes the check if $\Delta_b \leq t$. The algorithm always outputs a b such that ik_b passes the check. It outputs $\perp$ if both fail.

- If $\Delta_0 \leq t$ and $\Delta_1 \leq t$ return a random bit.
- If $\Delta_0 \leq t$, return 0.
- If $\Delta_1 \leq t$, return 1.
- Return $\perp$.

Fig. 4: Checking index keys: $\mathsf{ik\text{-}check}$ interacts with a copy of the adversarial verifier $\mathcal{A}$ initialised to state Q. Intuitively, the algorithm compares (for some fixed index key) the distribution resulting from generating a domain element via S_D and applying f, with the distribution resulting from generating a range element via S_R. If the index key is honestly generated, both distributions should be close. $\mathsf{ik\text{-}check}$ performs this comparison for both index keys the verifier $\mathcal{A}$ supplies in m_2. A key passes the check if both distributions appear sufficiently close, and $\mathsf{ik\text{-}check}$ can only output a key (or a bit indicating a key) that passes the check.

- Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b}\rangle, \widetilde{r}_{1-b}]$
- Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (\perp, \widetilde{q}_i, q_{1-b}, \perp, \perp, \perp)$.
- Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$.
- Send $m_3 := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$ to the verifier.
- Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

Output $\perp$

Success Stage:
If Inv *successfully inverts, we can use the output* q' *to produce a transcript using the third branch of* wi_3*. To do so, the simulator searches for a new challenge that also successfully inverts.*

For each $i \in [\kappa/\epsilon']$:

- Rewind the adversary to state Q (unless already in state Q).
- Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
- If Inv succeeds:
  - Rewind the adversary to state Q.
  - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
  - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
  - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
  - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (\perp, \perp, \perp, r_0, s, q')$
  - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$.
  - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

Output $\perp$

5.2 Proof of Theorem 2

Proof. Given the simulator described above, we will now prove the main theorem by contradiction: suppose there exists a distinguisher $\mathcal{D}$ that along with an adversarial verifier $\mathcal{A}$ distinguishes between experiments where the prover generates a proof using witness w versus an experiment where the proof is simulated, with advantage greater than ϵ, where $\epsilon = 1/poly(\kappa)$. We define $\epsilon' := \epsilon/7$ and consider a sequence of eight hybrid experiments, indexed by error parameter ϵ' where the first hybrid corresponds to the honest execution and the final hybrid corresponds to the simulated execution. $\mathcal{D}$ must necessarily distinguish some two consecutive hybrids in the sequence with advantage greater than $\epsilon' = \epsilon/7$. This leads to a contradiction, because we prove that the advantage of the distinguisher $\mathcal{D}$ for any two consecutive hybrids is always less than ϵ'.
In what follows, we let the random variable $\mathcal{D}(H_i^{\epsilon'})$ denote the output of the distinguisher upon receiving as input the view of the verifier in hybrid $H_i^{\epsilon'}$.

Hybrid $H_0^{\epsilon'}$:

This hybrid outputs the view of the verifier $\mathcal{A}$ when it interacts with an honest prover that generates a proof for x using witness w. The hybrid interacts with $\mathcal{A}$ acting as a prover for the protocol in the following manner:

- Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.

– Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
– Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
– Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
– Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
– Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails.
– Sample $q_0 \xleftarrow{R} S_D(\mathsf{ik}_0), q_1 \xleftarrow{R} S_D(\mathsf{ik}_1)$
– Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$
– Compute $\mathsf{chal}_0 = [F(\mathsf{ik}_0, q_0), \langle q_0, \widetilde{r}_0\rangle, \widetilde{r}_0]$ and $\mathsf{chal}_1 = [F(\mathsf{ik}_1, q_1), \langle q_1, \widetilde{r}_1\rangle, \widetilde{r}_1]$
– Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
– Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
– Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
– Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

Hybrid $H_1^{\epsilon'}$:

This hybrid differs from $H_0^{\epsilon'}$ in that it runs an additional algorithm $\mathsf{ik\text{-}check}$ once just after receiving the verifier's message. If $\mathsf{ik\text{-}check}$ outputs $\perp$, the hybrid aborts, else, it continues as in $H_0^{\epsilon'}$.

Interacting with $\mathcal{A}$ as a prover for the protocol, $H_1^{\epsilon'}$ performs the following:

– Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
– Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
– Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
– Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
– Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
– Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
– Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
– If $b = \perp$, return $\perp$
– Sample $q_0 \xleftarrow{R} S_D(\mathsf{ik}_0), q_1 \xleftarrow{R} S_D(\mathsf{ik}_1)$
– Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$
– Compute $\mathsf{chal}_0 = [F(\mathsf{ik}_0, q_0), \langle q_0, \widetilde{r}_0\rangle, \widetilde{r}_0]$ and $\mathsf{chal}_1 = [F(\mathsf{ik}_1, q_1), \langle q_1, \widetilde{r}_1\rangle, \widetilde{r}_1]$
– Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
– Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
– Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
– Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

Lemma 1. $\left|\Pr[\mathcal{D}(H_0^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_1^{\epsilon'}) = 1]\right| \leq \epsilon'$

Proof. (Overview) Note that the hybrids differ only if $\mathsf{ik\text{-}check}$ outputs $b = \perp$, and are identical conditioned on $b \in \{0,1\}$. Therefore, the probability of distinguishing the hybrids is bounded by the probability of $b = \perp$. We show that this probability is negligible in the full version of this paper, and omit the proof here due to space. Intuitively, this is because $\mathsf{ik\text{-}check}$ tests whether samples obtained by $f_{\mathsf{ik}}(S_D(\mathsf{ik}))$ and $S_R(\mathsf{ik})$ are indistinguishable to the adversary. ZAP ensures that with high probability atleast one of the index keys is sampled correctly, and by the enhancement property the samples will be indistinguishable. Therefore with high probability atleast one key will pass the test.

Hybrid $H_2^{\epsilon'}$:

This hybrid differs from $H_1^{\epsilon'}$ in that it runs $\mathsf{Inv}(f_{\mathsf{ik}_b}(q_b), \mathcal{Q}, b, \epsilon'/2)$ just before computing m_3 and ignores the output, here represented as following the same steps independent of the output.

Interacting with $\mathcal{A}$ as a prover for the protocol, $H_2^{\epsilon'}$ performs the following:

- Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
- Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
- Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
- Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
- Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
- Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
- Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
- If $b = \perp$, return $\perp$
- Sample $q_0 \xleftarrow{R} S_D(\mathsf{ik}_0), q_1 \xleftarrow{R} S_D(\mathsf{ik}_1)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(q_b), \mathcal{Q}, b, \epsilon'/2)$
- If Inv succeeds and outputs q' :
  - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$
  - Compute $\mathsf{chal}_0 = [F(\mathsf{ik}_0, q_0), \langle q_0, \widetilde{r}_0 \rangle, \widetilde{r}_0]$ and $\mathsf{chal}_1 = [F(\mathsf{ik}_1, q_1), \langle q_1, \widetilde{r}_1 \rangle, \widetilde{r}_1]$
  - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
  - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
  - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
- If Inv fails:
  - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$
  - Compute $\mathsf{chal}_0 = [F(\mathsf{ik}_0, q_0), \langle q_0, \widetilde{r}_0 \rangle, \widetilde{r}_0]$ and $\mathsf{chal}_1 = [F(\mathsf{ik}_1, q_1), \langle q_1, \widetilde{r}_1 \rangle, \widetilde{r}_1]$
  - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
  - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
  - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

Lemma 2. $\left|\Pr[\mathcal{D}(H_1^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_2^{\epsilon'}) = 1]\right| = 0$

Proof. The distributions of m_1, m_2, m_3 in $H_1^{\epsilon'}$ and $H_2^{\epsilon'}$ are identical because the output Inv is effectively ignored, therefore the adversary's view is identically distributed between both hybrids.

Hybrid $H_3^{\epsilon'}$:

This hybrid differs from $H_2^{\epsilon'}$ in that instead of computing the protocol execution using the same q_b for which Inv was called, it attempts to sample some $\widetilde{q}$ identically distributed to q_b given the output of Inv, and completes protocol execution using $\widetilde{q}$ if it succeeds.

Interacting with $\mathcal{A}$ as a prover for the protocol, $H_3^{\epsilon'}$ performs the following:

- Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
- Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
- Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
- Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
- Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
- Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
- Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
- If $b = \perp$, return $\perp$
- Sample $q_0 \xleftarrow{R} S_D(\mathsf{ik}_0), q_1 \xleftarrow{R} S_D(\mathsf{ik}_1)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(q_b), \mathcal{Q}, b, \epsilon'/2)$
- If Inv succeeds and outputs q', for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).
  - Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
  - If Inv succeeds:
    - Rewind the adversary to state Q.
    - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
    - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
    - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
    - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
    - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
    - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
    - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
- Else if Inv fails, for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).

- Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
- If Inv fails:
  * Rewind the adversary to state Q.
  * Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
  * Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
  * Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
  * Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
  * Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
  * Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  * Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

– Output $\perp$

Lemma 3. $\left|\Pr[\mathcal{D}(H_2^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_3^{\epsilon'}) = 1]\right| \leq \epsilon'$

Proof. Note that conditioned on successfully finding an appropriate $\widetilde{q}$, the adversary's view is identically distributed in both hybrids. The hybrid distinguishing probability is therefore bounded by the probability of failing to find $\widetilde{q}$.

For a fixed $(\mathcal{Q}, b)$, let p be defined as follows:

$$p(\mathcal{Q}, b) = \Pr_{\widetilde{q} \xleftarrow{R} S_D(\mathsf{ik}_b)} [\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}), \mathcal{Q}, b, \epsilon'/2) \text{ succeeds}]$$

We omit the parameters in the remaining discussion. Summing the probabilities of failing to find $\widetilde{q}$:

$$\left|\Pr[\mathcal{D}(H_2^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_3^{\epsilon'}) = 1]\right| \leq p(1-p)^{\kappa/\epsilon'} + (1-p)p^{\kappa/\epsilon'}$$

If $p \leq \epsilon'/2$, then:

$$p(1-p)^{\kappa/\epsilon'} + (1-p)p^{\kappa/\epsilon'} \leq \epsilon'/2 + (1-p)p^{\kappa/\epsilon'} \leq \epsilon'/2 + (\epsilon'/2)^{\kappa/\epsilon'} \leq \epsilon'$$

If $p > \epsilon'/2$, then:

$$p(1-p)^{\kappa/\epsilon'} + (1-p)p^{\kappa/\epsilon'} \leq \left(1 - \frac{\epsilon'}{2}\right)^{\kappa/\epsilon'} + (\epsilon'/2)^{\kappa/\epsilon'}$$

By the Taylor series expansion, we have:

$$\log\left(1 - \frac{\epsilon'}{2}\right)/\epsilon' \leq -\frac{1}{2}$$

which imples

$$\left(1-\frac{\epsilon'}{2}\right)^{\kappa/\epsilon'} = e^{\frac{\kappa}{\epsilon'}\log\left(1-\frac{\epsilon'}{2}\right)} \leq \frac{1}{e^{\kappa/2}}$$

Since $\epsilon' = 1/\mathsf{poly}(\kappa)$, for sufficiently large κ :

$$\left(1-\frac{\epsilon'}{2}\right)^{\kappa/\epsilon'} + (\epsilon'/2)^{\kappa/\epsilon'} \leq \frac{1}{e^{\kappa/2}} + \epsilon'/2 \leq \epsilon'$$

This completes the proof.

Hybrid $H_4^{\epsilon'}$:

This hybrid differs from $H_3^{\epsilon'}$ in that the adversary flips the hardcore bit for chal_b in the case where Inv fails.

Interacting with $\mathcal{A}$ as a prover for the protocol, $H_4^{\epsilon'}$ performs the following:

- Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
- Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
- Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
- Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
- Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
- Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
- Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
- If $b = \perp$, return $\perp$
- Sample $q_0 \xleftarrow{R} S_D(\mathsf{ik}_0), q_1 \xleftarrow{R} S_D(\mathsf{ik}_1)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(q_b), \mathcal{Q}, b, \epsilon'/2)$
- If Inv succeeds and outputs q', for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).
  - Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
  - If Inv succeeds:
    - Rewind the adversary to state Q.
    - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
    - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
    - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
    - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
    - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
    - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
    - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
- Else if Inv fails, for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).

- Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
- If Inv fails:
  * Rewind the adversary to state Q.
  * Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
  * Compute $\mathsf{chal}_b = \left[F(\mathsf{ik}_b, \widetilde{q}_i),\ \langle \widetilde{q}_i, \widetilde{r}_b \rangle \oplus 1\ , \widetilde{r}_b\right]$
  * Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
  * Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
  * Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
  * Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  * Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

– Output $\perp$

Lemma 4. $\left|\Pr[\mathcal{D}(H_3^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_4^{\epsilon'}) = 1]\right| \leq \epsilon'$

Proof. (Overview) Due to lack of space, we defer the proof to the full version. Intuitively, since the inversion algorithm fails for $\widetilde{q}_i$, the distinguisher does not distinguish between correct and incorrect hardcore bits. Since the only difference between hybrids is the hardcore bit, we show it is safe to send incorrect bits.

Hybrid $H_5^{\epsilon'}$:

This hybrid differs from $H_4^{\epsilon'}$ in that the adversary attempts to invert a randomly sampled $S_R(\mathsf{ik}_b)$ instead of $f_{\mathsf{ik}_b}(q_b)$. Since this change means that q_b is unused, it is no longer sampled. Interacting with $\mathcal{A}$ as a prover for the protocol, $H_5^{\epsilon'}$ performs the following:

– Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
– Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, compute $c = \mathsf{com}(r_0; s)$, compute the first message zap_1 of a ZAP, the first message wi_1 of a WI, and send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to the adversary.
– Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
– Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
– Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
– If $b = \perp$, return $\perp$
– Sample $q_b \xleftarrow{R} S_D(\mathsf{ik}_b)$
– Run $\mathsf{Inv}(\ S_R(\mathsf{ik}_b; \widehat{r})\ , \mathcal{Q}, b, \epsilon'/2)$ for $\widehat{r} \xleftarrow{R} \{0,1\}^\kappa$.
– If Inv succeeds and outputs q', for each $i \in [\kappa/\epsilon']$:
  • Rewind the adversary to state Q (unless already in state Q).
  • Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  • Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.

- If Inv succeeds:
  - Rewind the adversary to state Q.
  - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
  - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
  - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
  - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
  - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
  - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

– Else if Inv fails, for each $i \in [\kappa/\epsilon']$:
- Rewind the adversary to state Q (unless already in state Q).
- Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
- Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
- If Inv fails:
  - Rewind the adversary to state Q.
  - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
  - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle \oplus 1, \widetilde{r}_b]$
  - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
  - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
  - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
  - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

– Output $\perp$

Lemma 5. $\left|\Pr[\mathcal{D}(H_4^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_5^{\epsilon'}) = 1]\right| \leq \epsilon'$

Proof. Note that the behaviour of both hybrids for a given outcome of the first Inv call (success or failure) is identical. If a difference arises, it must be in the probability of Inv succeeding. Fix any $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$, where m_1 is parsed as $(c, \mathsf{zap}_1, \mathsf{wi}_1)$ and m_2 is parsed as $(r_1, \mathsf{ik}_1, \mathsf{ik}_2, \mathsf{wi}_2, \mathsf{zap}_2)$. Define predicates p_0, p_1, and p as:

$$p_0(b) = \Pr\left[\mathsf{Inv}(f_{\mathsf{ik}_b}(q_b), \mathcal{Q}, b, \epsilon'/2) \text{ succeeds.}\right]$$

$$p_1(b) = \Pr\left[\mathsf{Inv}(S_R(\mathsf{ik}_b), \mathcal{Q}, b, \epsilon'/2) \text{ succeeds.}\right]$$

$$p(b) = |p_0(b) - p_1(b)|$$

Further, define p as zero whenever $b = \perp$. If we let $\Pr[b]$ be the probability that $\mathsf{ik\text{-}check}_{\mathcal{Q}}$ outputs b in either hybrid:

$$\left|\Pr[\mathcal{D}(H_4^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_5^{\epsilon'}) = 1]\right| \leq \max_{\mathcal{Q}} \left(\sum_b \Pr[b] * p(b) \right)$$

where we consider $b \in \{0,1\}$ since p is zero if $b = \perp$. Let $\{X_i\}_{i\in[n']}$ be n' iid random variables that take values $\in \{0,1\}$, and $\Pr[X_i = 1] = p_1(m_1, m_2, Q, b)$. Similarly, let $\{Y_i\}_{i\in[n']}$ be n' iid random variables that take values $\in \{0,1\}$, and $\Pr[Y_i = 1] = p_0(m_1, m_2, Q, b)$. Let $Z = \sum_{i\in[n']} X_i - Y_i$. Then $\Pr[Z \leq t] = \Pr[\mathsf{ik}_b \text{ passes key-check}]$. Since (m_1, m_2, Q, b) are fixed for the following discussion, we omit the parameters from p_0, p_1, p. Note that $E[Z] = pn'$ and $Var[Z] = n'(Var[X] + Var[Y]) = n'p_1(1-p_1) + n'p_0(1-p_0) \leq 2n'$

Now, consider the case when $p \geq \epsilon'/2$. By applying Chebyshev's Inequality to Z we obtain:

$$\Pr\left[Z \leq E[Z] - k\sqrt{Var[Z]}\right] \leq 1/k^2$$

$E[Z] = n'p$ and $Var[Z] \leq 2n'$, therefore:

$$\Pr\left[Z \leq n'p - k\sqrt{2n'}\right] \leq 1/k^2$$

Since $p \geq \epsilon'/2$:

$$\Pr\left[Z \leq n'\epsilon'/2 - k\sqrt{2n'}\right] \leq 1/k^2$$

Setting $k = n'\epsilon'/\sqrt{32n'}$:

$$\Pr[Z \leq n'\epsilon'/4] = \Pr[Z \leq t] \leq \frac{32}{n'(\epsilon')^2}$$

Since $n' = (4/\epsilon')^3$:

$$\Pr[Z \leq t] \leq \epsilon'/2$$

$$\Pr[\mathsf{ik}_b \text{ passes key-check}] \leq \epsilon'/2$$

We use this result to show that the hybrid distinguishing probability is also bounded by ϵ'. If $p(b) \geq \epsilon'/2$, then probability of ik_b passing the check is no more than $\epsilon'/2$, which implies that $\Pr[b] \leq \epsilon'/2$. Therefore, $\Pr[b] * p(b) \leq \epsilon'/2$, which in turn implies $\sum_b \Pr[b] * p(b) \leq \epsilon'$. This completes the proof.

Hybrid $H_6^{\epsilon'}$:

This hybrid differs from $H_5^{\epsilon'}$ in that the adversary uses $r_0 \oplus r_1$ instead of fresh randomness to sample from $S_R(\mathsf{ik}_b)$.

Interacting with $\mathcal{A}$ as a prover for the protocol, $H_6^{\epsilon'}$ performs the following:

- Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
- Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
- Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
- Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
- Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.

– Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
– Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
– If $b = \perp$, return $\perp$
– Sample $q_b \xleftarrow{R} S_D(\mathsf{ik}_b)$
– Run $\mathsf{Inv}(\ S_R(\mathsf{ik}_b; r_0 \oplus r_1)\ , \mathcal{Q}, b, \epsilon'/2)$.
– If Inv succeeds and outputs q', for each $i \in [\kappa/\epsilon']$:
  • Rewind the adversary to state Q (unless already in state Q).
  • Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  • Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
  • If Inv succeeds:
    ∗ Rewind the adversary to state Q.
    ∗ Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
    ∗ Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
    ∗ Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
    ∗ Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
    ∗ Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
    ∗ Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
    ∗ Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
– Else if Inv fails, for each $i \in [\kappa/\epsilon']$:
  • Rewind the adversary to state Q (unless already in state Q).
  • Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  • Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
  • If Inv fails:
    ∗ Rewind the adversary to state Q.
    ∗ Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
    ∗ Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle \oplus 1, \widetilde{r}_b]$
    ∗ Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
    ∗ Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
    ∗ Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
    ∗ Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
    ∗ Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
– Output $\perp$

Lemma 6. $\left|\Pr[\mathcal{D}(H_5^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_6^{\epsilon'}) = 1]\right| \leq \epsilon'$

Proof. (Overview) Due to lack of space, we defer the proof to the full version. Intuitively, we switch from inverting a uniformly random string to inverting $r_0 \oplus r_1$. Since r_0 is also sampled uniformly, the adversary's view is identical apart from the commitment to r_0 in the first round. We therefore show that an adversary that distinguishes the hybrids must be breaking the hiding of the commitment scheme.

Hybrid $H_7^{\epsilon'}$:

This hybrid differs from $H_6^{\epsilon'}$ in that the hybrid uses the q' obtained by Inv to compute wi_3 by the third branch in the success case.

Interacting with $\mathcal{A}$ as a prover for the protocol, $H_7^{\epsilon'}$ performs the following:

- Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
- Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
- Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
- Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
- Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
- Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
- Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
- If $b = \perp$, return $\perp$
- Sample $q_b \xleftarrow{R} S_D(\mathsf{ik}_b)$
- Run $\mathsf{Inv}(S_R(\mathsf{ik}_b; r_1 \oplus r_2), \mathcal{Q}, b, \epsilon'/2)$.
- If Inv succeeds and outputs q', for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).
  - Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
  - If Inv succeeds:
    - Rewind the adversary to state Q.
    - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
    - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
    - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
    - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (\perp, \perp, \perp, r_0, s, q')$
    - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
    - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
    - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
- Else if Inv fails, for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).
  - Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
  - If Inv fails:
    - Rewind the adversary to state Q.
    - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
    - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle \oplus 1, \widetilde{r}_b]$
    - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
    - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (w, \perp, \perp, \perp, \perp, \perp)$
    - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
    - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.

* Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.

– Output $\perp$

Lemma 7. $\left|\Pr[\mathcal{D}(H_6^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_7^{\epsilon'}) = 1]\right| \leq \epsilon'$

Proof.

$$\Pr_{\substack{r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa, b_{\mathsf{wi}} \xleftarrow{R} \{0,1\} \\ R_1, R_2, R_3, \mathsf{WI}_3}} \left[b' = b_{\mathsf{wi}} \;\middle|\; \begin{array}{r} \mathsf{wi}_1 \leftarrow \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}}) \\ \mathsf{wi}_2, \{x_{\mathsf{wi},i}, w_{\mathsf{wi},i}\}_i, \tau \leftarrow R_1(\mathsf{wi}_1) \\ \{\mathsf{wi}_{3,i}\}_i \leftarrow \{\mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi},i}, w_{\mathsf{wi},i})\}_i \\ (x_{\mathsf{wi}}, w_0, w_1) \leftarrow R_2(\{\mathsf{wi}_{3,i}\}_i, \tau) \\ b' \leftarrow R_3(\mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{b_{\mathsf{wi}}})) \end{array} \right] \geq \frac{1}{2} + \frac{\epsilon'}{2}$$

Apart from the final construction of wi_3, WI_3 is always called during a call to Pred, which in turn is called during a call to Inv. This means that WI_3 is called for some $x'_{\mathsf{wi}} := (x', \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w'_{\mathsf{wi}} := (w', \perp^5)$. Here, x' and w' are chosen randomly each time, while $\mathsf{ik}_0, \mathsf{ik}_1, r_1$ and c are fixed by m_1 and m_2 before any WI_3 call is made. Only the distributions of chal_0 and chal_1 change between Pred calls. For any fixed b, the distributions of chal will be independent of the output of any Pred call. The only interdependence will be dependence on the bit b. This can be remedied by choosing two sets of inputs, one for $b = 0$ and one for $b = 1$. The outputs of one set may be later discarded depending on the value of b. Therefore, all inputs may be chosen together immediately after receiving m_2. We represent this set of WI_3 inputs as $S_{\mathsf{wi}} := \{x_{\mathsf{wi}_i}, w_{\mathsf{wi},i}\}_{i \in [\mathsf{poly}(\kappa)]}$

We build a reduction R that interacts with an external challenger for WI by running $H_6^{\epsilon'}$ with the following modifications:

1. Instead of computing the first message of WI, receive wi_1 from the external challenger.
2. Send wi_2 parsed from m_2 to the external verifier.
3. Choose the set S_{wi} as specified above and send to the external verifier immediately after sending wi_2.
4. Receive $P_{\mathsf{wi}} := \{\mathsf{wi}_{3,i}\}_{i \in [\mathsf{poly}(\kappa)]}$ from the challenger immediately after sending S_{wi}, consisting of third WI messages for each $(x_{\mathsf{wi},i}, w_{\mathsf{wi},i})$ pair in S_{wi}.
5. Proceed with hybrid execution until just before wi_3 is computed, replacing every WI_3 call with its corresponding $\mathsf{wi}_{3,i}$.
6. Compute w_{wi} as in $H_6^{\epsilon'}$ and label it w_0.
7. Compute w_{wi} as in $H_7^{\epsilon'}$ and label it w_1.
8. Send $(x_{\mathsf{wi}}, w_0, w_1)$ to the external challenger and receive wi_3 as response.
9. Continue with hybrid execution using wi_3 received from the challenger.
10. If the distinguisher returns 1, output 1. If not, or if any step fails, output 0.

The view of the adversarial verifier $\mathcal{A}$ in R is identical to the view in $H_6^{\epsilon'}$ when $b_{\mathsf{wi}} = 0$ and the view in $H_7^{\epsilon'}$ when $b_{\mathsf{wi}} = 1$. If we denote the event that R outputs $b' = b_{\mathsf{wi}}$ as R succeeding:

$$\Pr[R \text{ succeeds} | b_{\mathsf{wi}} = 1] = \Pr[\mathcal{D}(H_7^{\epsilon'}) = 1]$$

$$\Pr[R \text{ succeeds} | b_{\mathsf{wi}} = 0] = \Pr[\mathcal{D}(H_6^{\epsilon'}) \neq 1] = 1 - \Pr[\mathcal{D}(H_6^{\epsilon'}) = 1]$$

Since b_{wi} is chosen uniformly:

$$\Pr[R \text{ succeeds}] = \frac{1}{2}\left(1 + \left(\Pr[\mathcal{D}(H_7^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_6^{\epsilon'}) = 1]\right)\right) \geq \frac{1}{2} + \frac{\epsilon'}{2}$$

which contradicts the security of wi since $\epsilon'/2$ is non-negligible.

Hybrid $H_8^{\epsilon'}$:

This hybrid differs from $H_7^{\epsilon'}$ in that the hybrid uses the $\widetilde{q}$ for which Inv fails to compute wi_3 via the third branch in the failure case.

Interacting with $\mathcal{A}$ as a prover for the protocol, $H_8^{\epsilon'}$ performs the following:

- Sample $x, w \xleftarrow{R} \mathcal{X}_\kappa, \mathcal{W}_\kappa$.
- Sample $r_0 \xleftarrow{R} \{0,1\}^\kappa$, $s \xleftarrow{R} \{0,1\}^\kappa$, $r_{\mathsf{wi}} \xleftarrow{R} \{0,1\}^\kappa$
- Compute $c = \mathsf{com}(r_0; s)$, $\mathsf{zap}_1 \leftarrow \mathsf{ZAP}_1(1^\kappa)$, $\mathsf{wi}_1 \xleftarrow{R} \mathsf{WI}_1(1^\kappa; r_{\mathsf{wi}})$
- Send $m_1 = (c, \mathsf{zap}_1, \mathsf{wi}_1)$ to a freshly initialised instance of the adversary $\mathcal{A}$.
- Receive m_2, parsed as $(r_1, \mathsf{ik}_0, \mathsf{ik}_1, \mathsf{wi}_2, \mathsf{zap}_2)$.
- Verify that $\mathsf{ZAP}_{\mathsf{verify}}(\mathsf{zap}_1, \mathsf{zap}_2)$ accepts. Output $\perp$ if verification fails. Let the state of $\mathcal{A}$ at this point be Q.
- Define $\mathcal{Q} := (r_{\mathsf{wi}}, m_1, m_2, Q)$ and compute $b \xleftarrow{R} \mathsf{ik\text{-}check}_{\mathcal{Q}}$, where $b \in \{0, 1, \perp\}$
- If $b = \perp$, return $\perp$
- Sample $q_b \xleftarrow{R} S_D(\mathsf{ik}_b)$
- Run $\mathsf{Inv}(S_R(\mathsf{ik}_b; r_1 \oplus r_2), \mathcal{Q}, b, \epsilon'/2)$.
- If Inv succeeds and outputs q', for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).
  - Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.
  - If Inv succeeds:
    - Rewind the adversary to state Q.
    - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
    - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle, \widetilde{r}_b]$
    - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
    - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$ and $w_{\mathsf{wi}} := (\perp, \perp, \perp, r_0, s, q')$
    - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
    - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
    - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
- Else if Inv fails, for each $i \in [\kappa/\epsilon']$:
  - Rewind the adversary to state Q (unless already in state Q).
  - Sample $\widetilde{q}_i \xleftarrow{R} S_D(\mathsf{ik}_b)$
  - Run $\mathsf{Inv}(f_{\mathsf{ik}_b}(\widetilde{q}_i), \mathcal{Q}, b, \epsilon'/2)$.

- If Inv fails:
  - Rewind the adversary to state Q.
  - Sample $\widetilde{r}_0, \widetilde{r}_1 \xleftarrow{R} \{0,1\}^\kappa$, $q_{1-b} \xleftarrow{R} S_D(\mathsf{ik}_{1-b})$.
  - Compute $\mathsf{chal}_b = [F(\mathsf{ik}_b, \widetilde{q}_i), \langle \widetilde{q}_i, \widetilde{r}_b \rangle \oplus 1, \widetilde{r}_b]$
  - Compute $\mathsf{chal}_{1-b} = [F(\mathsf{ik}_{1-b}, q_{1-b}), \langle q_{1-b}, \widetilde{r}_{1-b} \rangle, \widetilde{r}_{1-b}]$
  - Set $x_{\mathsf{wi}} := (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{ik}_0, \mathsf{ik}_1, r_1, c)$
  - If $b = 0$:
    - Set $w_{\mathsf{wi}} := (\bot, \widetilde{q}_i, q_{1-b}, \bot, \bot, \bot)$
  - If $b = 1$:
    - Set $w_{\mathsf{wi}} := (\bot, q_{1-b}, \widetilde{q}_i, \bot, \bot, \bot)$
  - Compute $\mathsf{wi}_3 \leftarrow \mathsf{WI}_3(r_{\mathsf{wi}}, \mathsf{wi}_2, x_{\mathsf{wi}}, w_{\mathsf{wi}})$
  - Send $m_3 = (x, \mathsf{chal}_0, \mathsf{chal}_1, \mathsf{wi}_3)$.
  - Output the view of the verifier for the current session, i.e. for m_1, m_2, m_3, and halt execution.
- Output $\bot$

Lemma 8. $\left|\Pr[\mathcal{D}(H_8^{\epsilon'}) = 1] - \Pr[\mathcal{D}(H_7^{\epsilon'}) = 1]\right| \leq \epsilon'$

Proof. The proof proceeds identically to the proof of Lemma 7, except that in R, w_0 is computed as wi_3 in $H_7^{\epsilon'}$ instead of as in $H_6^{\epsilon'}$ and w_1 is computed as wi_3 in $H_8^{\epsilon'}$ instead of as in $H_7^{\epsilon'}$.

The view of the adversary in the final hybrid $H_8^{\epsilon'}$ is identically distributed to the simulated view of the adversary. Since $\mathcal{D}$ is unable to distinguish any two consecutive hybrids with probability greater than ϵ', it cannot distinguish the view of the adversary in an honest execution from the simulated view with probability greater than ϵ.

5.3 Argument of Knowledge property

Theorem 3. *Assuming enhanced injective trapdoor functions secure against PPT adversaries, the protocol in Figure 1 is an argument of knowledge.*

Proof. (Overview) Due to lack of space, we defer the proof to the full version. Intuitively, the extractor uses the argument of knowledge property of WI to extract a witness. Since an honest verifier will check challenge bits, this will either yield a witness for $x \in L$ or an inverse of $r_0 \oplus r_1$, which we show implies breaking one-wayness of f. One subtlety is that the inverse may be with respect to either index key. We show that we may obtain an inverse with respect to the desired index key by swapping the order of keys, which is undetectable by the witness indistinguishability of ZAP.

Acknowledgments

D. Khurana and K. Tomer were supported in part by NSF CAREER CNS-2238718, DARPA SIEVE and a gift from Visa Research. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024. G. Malavolta was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972.

References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (May 2001)
2. Badrinarayanan, S., Goyal, V., Jain, A., Khurana, D., Sahai, A.: Round optimal concurrent MPC via strong simulation. IACR Cryptology ePrint Archive **2017**, 597 (2017)
3. Bitansky, N., Freizeit, S.: Statistically sender-private OT from LPN and derandomization. CRYPTO (2022)
4. Bitansky, N., Khurana, D., Paneth, O.: Weak zero-knowledge beyond the black-box barrier. SIAM Journal on Computing Special Section for STOC 2019 pp. STOC19–156–STOC19–199 (2022)
5. Brakerski, Z., Döttling, N.: Two-message statistically sender-private OT from LWE. In: Beimel, A., Dziembowski, S. (eds.) Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11240, pp. 370–390. Springer (2018)
6. Deng, Y.: Individual simulations. In: ASIACRYPT 2020, Part III. pp. 805–836. LNCS, Springer, Heidelberg (Dec 2020)
7. Döttling, N., Garg, S., Goyal, V., Malavolta, G.: Laconic conditional disclosure of secrets and applications. In: FOCS. pp. 661–685. IEEE Computer Society (2019)
8. Döttling, N., Garg, S., Hajiabadi, M., Masny, D., Wichs, D.: Two-round oblivious transfer from CDH or LPN. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 12106, pp. 768–797. Springer (2020)
9. Dwork, C., Naor, M.: Zaps and their applications. SIAM J. Comput. **36**(6), 1513–1543 (2007)
10. Dwork, C., Naor, M., Reingold, O., Stockmeyer, L.J.: Magic functions. J. ACM **50**(6), 852–921 (2003)
11. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: STOC. pp. 308–317. IEEE Computer Society (1990)
12. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA. pp. 416–426. ACM (1990)
13. Goldreich, O.: The Foundations of Cryptography - Volume 1, Basic Techniques. Cambridge University Press (2001)
14. Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press (2004)

15. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: Johnson, D.S. (ed.) Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA. pp. 25–32. ACM (1989)
16. Goldreich, O., Rothblum, R.D.: Enhancements of trapdoor permutations. Journal of Cryptology **26**, 484–512 (2013)
17. Goyal, V., Richelson, S.: Non-malleable commitments using goldreich-levin list decoding. In: Zuckerman, D. (ed.) 60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019. pp. 686–699. IEEE Computer Society (2019)
18. Halevi, S., Kalai, Y.T.: Smooth projective hashing and two-message oblivious transfer. J. Cryptology **25**(1), 158–193 (2012)
19. Jain, A., Kalai, Y.T., Khurana, D., Rothblum, R.: Distinguisher-dependent simulation in two rounds and its applications. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 158–189. Springer (2017)
20. Kalai, Y.T., Khurana, D., Sahai, A.: Statistical witness indistinguishability (and more) in two messages. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 34–65. Springer (2018)
21. Khurana, D.: Round optimal concurrent non-malleability from polynomial hardness. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10678, pp. 139–171. Springer (2017)
22. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA. pp. 448–457 (2001)
23. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: 22nd ACM STOC. pp. 427–437. ACM Press (May 1990)