Leakage-Resilient Zero Knowledge*

Sanjam Garg Abhishek Jain Amit Sahai

UCLA

Abstract

In this paper, we initiate a study of zero knowledge proof systems in the presence of side-channel attacks. Specifically, we consider a setting where a cheating verifier is allowed to obtain arbitrary bounded leakage on the *entire state* (*including the witness and the random coins*) of the prover *during the entire protocol execution*. We formalize a meaningful definition of *leakage-resilient zero knowledge* (LR-ZK) proof system, that intuitively guarantees that *the protocol does not yield anything beyond the validity of the statement and the leakage obtained by the verifier*.

We give a construction of LR-ZK interactive proof system based on standard general assumptions. To the best of our knowledge, this is the first instance of a cryptographic *interactive protocol* where the adversary is allowed to perform leakage attacks during the protocol execution on the *entire state* of honest party (in contrast, prior work only considered leakage *prior* to the protocol execution, or very limited leakage *during* the protocol execution). Next, we give an LR-NIZK proof system based on standard number-theoretic assumptions.

Finally, we demonstrate the usefulness of our notions by giving two concrete applications:

- We initiate a new line of research to relax the assumption on the "tamper-proofness" of hardware tokens used in the design of various cryptographic protocols. In particular, by making use of our LR-ZK proof system (and the framework of Lin et al. [STOC'09]), we give a construction of a universally composable multiparty computation protocol in the *leaky token model* (where an adversary in possession of a token is allowed to obtain arbitrary bounded leakage on the *entire state* of the token) based on standard general assumptions.
- Next, we extend our notion of LR-NIZK to include the property of *simulation extractability*. Then, by adapting the approach of Katz and Vaikuntanathan [Asiacrypt'09] and Dodis et al. [Asiacrypt'10, FOCS'10] (for constructing "standard" leakage-resilient signature schemes) to the setting of "full-leakage" (where the adversary can leak on the *entire state* as opposed to only the secret key), we obtain simple, generic constructions of *fully* leakage-resilient (FLR) signatures in the bounded leakage model as well as the continual leakage model. In contrast to the recent constructions of FLR signature schemes, our scheme is also secure in the "noisy leakage" model. We supplement our result by showing that a simulation-extractable LR-NIZK proof system is implied by the UC-NIZK of Groth et al. [Eurocrypt'06].

*This is a preliminary full version of our CRYPTO'11 paper.

1 Introduction

Zero knowledge proof systems, introduced in the seminal work of Goldwasser, Micali and Rackoff [GMR85], have proven fundamental to cryptography. Very briefly, a zero knowledge proof system is an interactive proof between two parties – a prover, and a verifier – with the remarkable property that the verifier does not learn anything beyond the validity of the statement being proved. Subsequent to their introduction, zero knowledge proofs have been studied in various adversarial settings such as concurrency attacks [DNS98], malleability attacks [DDN00], reset attacks [CGGM00], to list a few, with very successful results. Over the years, zero knowledge proofs (and its various strengthened notions) have turned to be extremely useful, finding numerous applications in the design of various cryptographic protocols.

We note that the standard definition of zero knowledge proofs, like most classical security notions, assumes that an adversary is given only black-box access to the honest party algorithms. Unfortunately, over the last two decades, it has become increasingly evident that such an assumption may be unrealistic when arguing security in the real world where the physical implementation (e.g. on a smart card or a hardware token) of an algorithm is under attack. Motivated by such a scenario, in this paper, we initiate a study of zero knowledge proof systems in the presence of *side-channel attacks* [Koc96, AK96, QS01, GMO01, OST06, HSH+08].Specifically, we study zero knowledge proofs in the intriguing setting where a cheating verifier, in addition to receiving a proof of some statement, is able to obtain arbitrary bounded leakage on the *entire state* (*including the witness and the random coins*) of the prover *during the entire protocol execution*. We note that while there has been an extensive amount of research work on leakage-resilient cryptography in the past few years, to the best of our knowledge, almost all prior work has either been on leakage resilient *primitives* such as encryption and signature schemes [DP08, AGV09, Pie09, DKL09, ADW09a, NS09, KV09, DGK+10, FKPR10, ADN+10, KP10, BKKV10, DHLW10a, DHLW10b, LRW11, MTVY11, BSW11, LLW11], or leakage-resilient (and tamper-resilient) *devices* [ISW03, IPSW06, FRR+10, Ajt11], while very limited effort has been dedicated towards constructing leakage-resilient *interactive protocols*. To the best of our knowledge, the recent works on correlation extractors [IKOS09], and leakage-resilient identification and authenticated key agreement protocols [ADW09a, DHLW10b, DHLW10a] come closest to being considered in the latter category. However, we stress that in all these works, either leakage attacks are allowed only *prior* to the protocol execution, or very limited leakage is allowed *during* the protocol execution; in contrast, we consider the setting where the adversary can obtain leakage on the *entire state* of the honest party during the protocol execution.

We find it imperative to stress that handling leakage attacks on interactive protocols can be particularly challenging. On the one hand, for the leakage attacks to be meaningful, we would want to allow leakage on the secret state of the protocol participants. However, the state of a party typically includes a secret value (witness and random coins of the prover in the case of zero knowledge proofs) and any leakage on that secret value might immediately violate a security property (e.g., the zero knowledge property) of the protocol. Then, coming back to setting of zero knowledge proofs, it is not immediately clear how to even define "leakage-resilient zero knowledge."

How to define Leakage-Resilient Zero Knowledge? One possibility is to pursue an assumption such as *only computation leaks information* [MR04] (i.e., assuming that there is no leakage in the absence of computation). While this is a valuable and interesting approach, we note that this assumption is often problematic (e.g. cold-boot attacks [HSH+08]). In our work here, therefore, we do *not* make any such assumption. We seek a general definition maximizing the potential applicability of that definition to different application scenarios.

Another possibility is to allow a "leakage-free pre-processing phase" prior to the actual protocol execution, in an attempt to render the leakage attacks during the protocol useless. We note, however,

that allowing pre-processing would limit the applicability of our notion. In particular, such a definition would be problematic for scenarios where the statement to be proven is generated "online" (thereby eliminating the possibility of pre-processing the witness "safely"). Furthermore, we give strong evidence that such an approach is unlikely to yield better guarantees than what we are able to achieve (see Section C for further discussion on this issue).

Indeed, our goal is to obtain a meaningful and appropriate definition of zero knowledge in the model where an adversarial verifier can obtain leakage on any content (state) of the prover machine at any time. We do not consider any "leakage-free" time-period; in particular, any pre-processing phase is subject to leakage as well. However, in such a setting, it is important to note that since the adversary could simply choose to leak on the *witness* (and no other prover state), the zero knowledge *simulator* must be able to obtain similar amount of leakage in order to perform correct simulation. We shall see that even with this limitation, our notion turns out to be both quite nontrivial to obtain and very useful in application scenarios.

Our Definition – Informally. To this end, we consider a definition of leakage-resilient zero knowledge that provides the intuitive guarantee that *the protocol does not yield anything beyond the validity of the statement and the leakage obtained by the adversary.* In other words, whatever an adversary "learns" from the protocol (with leakage) should be no more than what she can learn from only the leakage without running the protocol. To formalize the above intuition, as a first step, we consider a *leakage oracle* that gets as private input the witness of the honest prover; the zero knowledge simulator is then given access to such a leakage oracle. More concretely, we consider a parameter λ, and say that an interactive proof system is λ-*leakage-resilient zero knowledge* (LR-ZK) if for every cheating verifier, there exists a simulator with access to a leakage oracle (that gets the honest prover's witness as private input) that outputs a view of the verifier (indistinguishable from the real execution), with the following requirement. Let ℓ bits be an upper bound on the total amount of leakage obtained by the adversarial verifier. Then the simulator is allowed to obtain at most $\lambda \cdot \ell$ bits of leakage. (In Section B, we show that constructing an LR-ZK proof system with $\lambda < 1$ is in fact impossible.)

Applications of Our Definition. Now that we have a definition for LR-ZK proof system, one may question how meaningful it is. As we now discuss, the above definition indeed turns out to be very useful. Intuitively, our definition is appropriate for a scenario where a leakage-resilient primitive A is being used in conjunction with a zero knowledge proof system (where the proof system is used to prove some statement about A), in the design of another cryptographic protocol B. The reason for this is that our definition of LR-ZK allows us to directly reduce the leakage-resilience property of B on the leakage-resilience property of A.

As an application of our LR-ZK interactive proof system, we first construct a universally composable (UC) multiparty computation protocol in the *leaky token model* (which is a relaxation of the model of Katz [Kat07] in that a malicious token user is now allowed to leak arbitrary bounded information on the *entire state* of the token). Very briefly, we use *leakage-resilient hard relations* [DHLW10b] and hardware tokens that implement the prover algorithm of our LR-ZK proof system where we prove the validity of an instance of the hard relation; then the leakage on the state of the token can be easily "reduced" to leakage on (the witness corresponding to) an instance of the hard relation.

Next, we are able to extend the notion of LR-ZK to the non-interactive setting in a natural way. Then, as an application of LR-NIZKs, we give generic constructions of *fully* leakage-resilient (FLR) signature schemes (where leakage is allowed on the *entire state* as opposed to only the secret key). Very briefly, we use leakage-resilient hard relations in conjunction with "simulation-extractable" LR-NIZKs (see below); we are then able to reduce the leakage-resilience property of the signature scheme to that of the hard relation. We now summarize our results.

1.1 Our Results

We first study the possibility of constructing leakage-resilient zero knowledge protocols and obtain the following results:

- We construct a $(1+\epsilon)$-leakage-resilient zero knowledge interactive proof system (where ϵ is any positive constant) based on standard general assumptions (specifically, the existence of a statistically hiding commitment scheme that is public-coin w.r.t. the receiver). To the best of our knowledge, this is the first instance of a cryptographic *interactive protocol* where an adversary is allowed to obtain arbitrary bounded leakage on the *entire state* of the honest parties *during* the protocol execution.

- Next, we consider the non-interactive setting and show that any NIZK proof system with *honest prover state reconstruction* property [GOS06] is an LR-NIZK proof system for $\lambda = 1$. As a corollary, we obtain an LR-NIZK proof system from [GOS06] based on the decisional linear assumption.

We supplement our above positive results by proving the impossibility of constructing an LR-ZK proof (or argument) system for $\lambda < 1$. Then, as applications of leakage-resilient zero knowledge, we obtain the following results:

- We initiate a new line of research to relax the assumption on the "tamper-proofness" of hardware tokens used in the design of various cryptographic protocols. In particular, assuming semi-honest oblivious transfer, we give a construction of a universally composable (UC) multiparty computation protocol in the *leaky token model*, where the token user is allowed to obtain arbitrary bounded leakage on the *entire state* of the token. We stress that all prior works on designing cryptographic protocols using hardware tokens, including the work on UC secure computation [Kat07, CGS08, MS08, DNW09], made the implicit assumption that the tokens are completely leakage-resilient.

- Next, we extend the notion of leakage-resilient NIZKs to incorporate the property of *simulation-extractability* [Sah99, DDO$^+$01] (also see [PR05] in the context of interactive proofs), in particular, the "true" variant [DHLW10b]. We are then able to adapt the approach of Katz and Vaikuntanathan [KV09], and in particular, Dodis et al [DHLW10b, DHLW10a] (who use a leakage-resilient hard relation in conjunction with a true simulation-extractable NIZK argument system to construct leakage-resilient signatures) to the setting of *full leakage*. As a result, we obtain simple, generic constructions of *fully* leakage-resilient signature schemes in the bounded leakage model as well as the continual leakage model. Similar to [DHLW10b, DHLW10a], our signature scheme inherits the leakage-resilience properties (and the leakage bounds) of the hard relation used in its construction.[1] In contrast to the recent constructions of FLR signature schemes by [MTVY11, BSW11, LLW11] in the standard model[2], our scheme is also secure in the *noisy leakage model* [NS09]. We supplement our result by showing that a true simulation-extractable leakage-resilient NIZK argument system is implied by the UC-NIZK of Groth et al. [GOS06], which can be based on the decisional linear assumption.

[1] Specifically, our signature scheme is fully leakage-resilient (FLR) in the bounded (resp., continual) leakage model if the hard relation is leakage-resilient in the bounded (resp., continual) leakage model.As such, if we use the key pairs from the encryption scheme of Lewko et al [LLW11] as a hard relation, then our signature scheme can tolerate leakage during the update process as well.

[2] Earlier, FLR signature schemes were constructed either only in the random oracle model [ADW09a, DHLW10b, BKKV10], or were only "one-time" [KV09]

We study two more questions which are very closely related to the setting of leakage-resilient zero knowledge. First, we consider the scenario in which a malicious prover can obtain arbitrary leakage on the random coins of the verifier during the protocol execution. The question that we investigate is whether it is possible to construct interactive proofs that remain sound even in such a scenario. We refer to such proofs as *leakage-sound proofs*. Secondly, we consider the question of constructing an interactive proof system that *simultaneously* satisfies the two notions of leakage-soundness (c.f. Definition 12) and leakage-resilient zero knowledge (c.f. Definition 8). We call such an interactive proof system *simultaneous leakage-resilient zero knowledge*. We obtain positive results for both these settings. We refer the reader to Section 7 for details.

1.2 Our Techniques

We now briefly discuss the main techniques used to obtain our positive results on leakage-resilient zero knowledge proof systems. Recall that our goal is to realize a definition where a cheating verifier does not learn anything from the protocol beyond the validity of the statement and the leakage information obtained from the prover. Further, recall that in our definition, simulator is given access to a leakage oracle that gets the honest prover's witness as private input and accepts leakage queries on the witness string. (In contrast, the verifier is allowed to make leakage queries on the entire state, including the witness and the random coins used by the prover thus far in the protocol execution.) Then, during the simulation, on receiving a leakage query from the verifier, our simulator attempts to convert it into a "valid" query to the leakage oracle. Now, note that the simulator may be cheating in the protocol execution (which is typically the case since it does not possess a valid witness); then, since the verifier can arbitrarily leak on both the witness and the random coins (which completely determine the actions of the prover thus far), at every point in the protocol execution, the simulator must find a way to "explain its actions so far". Note that this is reminiscent of *adaptive security* [Bea96, CFGN96, CLOS02, LZ09] in the context of secure computation protocols. We stress, however, that adaptive security does not suffice to achieve the property of leakage-resilient zero knowledge in the interactive proofs setting, as we explain below.

Recall that the notion of adaptive security corresponds to the setting where an adversary is allowed to corrupt parties *during* the protocol execution (as opposed to static corruption, where the parties can only be corrupted *before* the protocol begins). Once a party is corrupted, the adversary learns the entire state (including the input and random coins) of that party. The adversary may choose to corrupt several parties (in the case of multi-party protocols) throughout the course of the protocol. The notion of adaptive security guarantees security for the remaining uncorrupted parties.

While adaptive corruption itself is not our focus, note that in our model, a cheating verifier may obtain leakage on the prover's state at *several points* during the protocol execution. Furthermore, the honest prover may not even be aware as to what was leaked. Our goal is to guarantee that the adversary does not learn anything beyond the leaked information. Then, in order to provide such a guarantee, note that our simulator must *continue to simulate the prover even after leakage happens*, in a way that is consistent with the leaked information even though it does not know the prover's witness or what information was leaked. In contrast, the simulator for adaptively secure protocols does *not* need to simulate a party once it is corrupted.[3] In summary, we wish to guarantee some security for the honest party even *after* leakage happens, while adaptive security does not provide any such guarantees. We stress that this difference is crucial, and explains why known techniques for achieving adaptive security do not suffice for our purposes. Nevertheless, as we explain below, adaptive security serves as a good starting point for our purpose.

Recall that the main challenge in the setting of adaptive security is that whenever an adversary chooses to corrupt a party, the simulator must be able to explain its random coins, in a way that

[3]Indeed, for this reason, known adaptively secure ZK protocols are not leakage-resilient.

is consistent with the party's input and the messages it generated so far in the protocol. The main technique for overcoming this challenge is to allow the simulator to *equivocate*. For our purposes, we will also make use of equivocation so that the leakage queries can be answered correctly by the simulator. However, since our simulator would need to simulate the prover even after leakage happens (without the knowledge of the prover's witness or the information that was leaked), we do not want this equivocation to interfere with the simulation of prover's messages. In other words we want to be able to simulate the prover's messages independent of what information is being leaked but still remain consistent with it. Our solution is to have two separate and independent ways of cheating at the simulator's disposal. It will use one way to cheat in the protocol messages and the second way is reserved for answering the leakage queries correctly. Furthermore, we would need to make sure that the simulator does not "step on its own toes" when using the two ways to cheat *simultaneously*.

We now briefly discuss the actual construction of our protocol in order to illustrate the above ideas. We recall two well-known ways of constructing constant-round zero knowledge protocols – the Feige-Shamir [FS89] approach of using equivocal commitments (also used in adaptive security), and the Goldreich-Kahan [GK96] approach of requiring the verifier to commit to its challenges in advance. Now, armed with the intuition that our simulator will need two separate ways of cheating, we use both the above techniques *together*. Our protocol roughly consists of two phases: in the first phase, the verifier commits to a challenge string using a standard challenge-response based extractable commitment scheme (in a manner similar to [Ros04]); in the second phase, we execute the Blum-Hamiltonicity protocol instantiated with an equivocal commitment scheme. While leakage during the first phase can be handled easily by our simulator, handling leakage during the second phase makes use of the ideas discussed above.

Unfortunately, although the above construction seems to satisfy most of our requirements, it fails on the following account. Recall that our goal is to obtain a leakage-resilient zero knowledge protocol with nearly optimal precision (i.e., $\lambda = 1+\epsilon$) with respect to the leakage queries of the simulator. Now note that in the above construction, the simulator would need to extract the verifier's challenge in the first phase by means of rewinding before proceeding to phase two of the protocol. Then, depending upon the verifier's behavior, the simulator may need to perform several rewinds in order to succeed in extraction. Now, note that a cheating verifier may be able to make a *different* leakage query during each rewind, thus forcing our simulator to make a new query as well to its leakage oracle. As a result, depending upon the number of such rewinds, the total leakage obtained by the simulator may potentially become a polynomial factor of the leakage obtained by the adversary in a real execution.

In order to obtain a precision in the leakage queries of the simulator, we borrow techniques from the work on *precise zero knowledge* pioneered by Micali and Pass [MP06]. We remark that in the context of precise ZK, (for fundamental reasons of modeling) it is typically not possible to obtain a precision of almost 1. In our case, however, we are able to achieve a precision of $\lambda = 1 + \epsilon$ (where ϵ is any positive constant) with respect to the leakage queries of the simulator.

Finally, we note that in the case of non-interactive zero knowledge, since the simulator does not need to simulate any "future messages" after the leakage, we are indeed able to show that an adaptively secure NIZK is also a leakage-resilient NIZK. Specifically, we show that any NIZK with *honest prover state reconstruction* property, as defined by Groth et al. [GOS06] (in the context of adaptive security), is also a leakage-resilient NIZK with $\lambda = 1$.

Roadmap. We start by recalling some basic definitions in Section 2. Then in Section 3 we introduce the notion of leakage-resilient zero knowledge protocols and give a concrete construction of a leakage-resilient zero knowledge proof system. Next we extend our results to the non-interactive setting in Section 4. Finally, as applications for our results we give a construction of Universally Composable secure protocols with leaky tokens (Section 5) and a construction of a fully leakage-resilient signature scheme (Section 6). We conclude with a note on leakage soundness in Section 7.

2 Preliminaries

2.1 Basic Definitions: Interactive Case

We first recall the standard definitions of interactive proofs and zero knowledge [GMR85]. For convenience, we will follow the notation and presentation of [PR05]. Let P (called the *prover*) and V (called the *verifier*) denote a pair of interactive Turing machines that are running a protocol with each other on common input x. Throughout our text, we will always assume V to be a polynomial-time machine. Let $\langle P, V\rangle(x)$ be the random variable representing the output of V at the end of the protocol. If the machine P is polynomial-time, it is assumed that it has a private input w.

Definition 1 (Interactive proof system) *A pair of interactive Turing machines $\langle P, V\rangle$ is called an* interactive proof system *for a language $\mathcal{L}$ if the following two conditions hold:*

- Completeness: *For every $x \in \mathcal{L}$,*

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

- Soundness: *For every $x \notin \mathcal{L}$, and every interactive Turing machine P^*,*

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

If the soundness condition in the above definition is valid only against PPT Turing machines, then we say that $\langle P, V\rangle$ is an *argument* system.

Zero Knowledge. An interactive proof $\langle P, V\rangle$ is said to be *zero-knowledge* if, informally speaking, the verifier V learns nothing beyond the validity of the statement being proved. This intuition is formalized by requiring that the view of every probabilistic polynomial-time (PPT) cheating verifier V^*, represented by $\text{view}_{V^*}(x, z)$, generated as a result of its interaction with P can be "simulated" by a PPT machine $\mathcal{S}$ (referred to as the *simulator*). Here, the verifier's view consists of the common input x, its random tape, and the sequence of prover messages that it receives during the protocol execution. The auxiliary input of V^* and $\mathcal{S}$ is denoted by $z \in \{0,1\}^*$.

Definition 2 (Zero knowledge) *An interactive proof system $\langle P, V\rangle$ for a language $\mathcal{L}$ is said to be* zero knowledge *if for every PPT verifier V^*, there exists a PPT algorithm $\mathcal{S}$ such that for every $x \in \mathcal{L}$, every $z \in \{0,1\}^*$, $\text{view}_{V^*}(x, z)$ and $\mathcal{S}(x, z)$ are computationally indistinguishable.*

One can consider stronger variants of zero knowledge where the output of $\mathcal{S}$ is statistically close (or identical) to the verifier's view. In this paper, unless otherwise specified, we will focus on the computational variant only.

2.2 Basic Definitions: Non-Interactive Case

Here we recall the standard definition of non-interactive zero knowledge (NIZK) proof systems. For convenience, we will follow the notation and presentation of [GOS06].

Let $\mathcal{R}$ be an efficiently computable relation that consists of pairs (x, w), where x is called the statement and w is the witness. Let $\mathcal{L}$ denote the language consisting of statements in $\mathcal{R}$. A non-interactive proof system for a language $\mathcal{L}$ consists of a setup algorithm K, a prover P and a verifier V. The setup algorithm K generates a common reference string σ. The prover P takes as input (σ, x, w) and checks whether $(x, w) \in \mathcal{R}$; if so, it produces a proof string π, else it outputs `fail`. The verifier V takes as input (σ, x, π) and outputs 1 if the proof is valid, and 0 otherwise.

Definition 3 (Non-interactive proof system) *A tuple of algorithms (K, P, V) is called a* non-interactive proof system *for a language $\mathcal{L}$ with a* PPT *relation $\mathcal{R}$ if the following two conditions hold:*

- *Completeness: For all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k); (x, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow P(\sigma, x, w) : V(\sigma, x, \pi) = 1 \text{ if } (x, w) \in \mathcal{R}] \geq 1 - \text{negl}(k)$$

- *Soundness: For all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k); (x, \pi) \leftarrow \mathcal{A}(\sigma) : V(\sigma, x, \pi) = 1 \text{ if } x \notin \mathcal{L}] \leq \text{negl}(k)$$

If the soundness condition holds only against PPT adversaries, then we say that (K, P, V) is a non-interactive *argument* system.

Definition 4 (Zero Knowledge) *A non-interactive proof system (K, P, V) for a relation $\mathcal{R}$ is said to be* zero knowledge *if there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k) : \mathcal{A}^{P(\sigma,\cdot,\cdot)}(\sigma) = 1] \stackrel{c}{\equiv} \Pr[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^k) : \mathcal{A}^{\mathcal{S}'(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1],$$

where $\mathcal{S}'(\sigma, \tau, x, w) = \mathcal{S}_2(\sigma, \tau, x)$ if $(x, w) \in \mathcal{R}$ and outputs `fail` *otherwise.*

We now state an extension of the zero knowledge property, called *honest prover state reconstruction*, that is central to our positive result on leakage-resilient NIZK. We recall the notion as defined by Groth, Ostrovsky and Sahai [GOS06].

Definition 5 (Honest prover state reconstruction) *We say that a non-interactive proof system (K, P, V) for a relation $\mathcal{R}$ has* honest prover state reconstruction *if there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that for all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k) : \mathcal{A}^{PR(\sigma,\cdot,\cdot)}(\sigma) = 1] \stackrel{c}{\equiv} \Pr[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^k) : \mathcal{A}^{SR(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1],$$

where $PR(\sigma, x, w)$ computes $r \leftarrow \{0,1\}^{\ell_P(k)}$; $\pi \leftarrow P(\sigma, x, w; r)$ and returns $(\pi, w, r,)$ and $SR(\sigma, \tau, x, w)$ computes $\rho \leftarrow \{0,1\}^{\ell_S(k)}$; $\pi \leftarrow \mathcal{S}_2(\sigma, \tau, x; \rho)$; $r \leftarrow \mathcal{S}_3(\sigma, \tau, x, w, \rho)$ and returns (π, w, r); both of the oracles outputting `fail` *if $(x, w) \notin \mathcal{R}$.*

2.3 Leakage-Resilient Primitives

Here we recall the notion of leakage-resilient hard relations as defined by Dodis, Haralambiev, Lopez-Alt, Wichs [DHLW10b].

To model leakage attacks, the adversary is given access to a *leakage oracle*, which she can adaptively access to learn leakage on the secret value. A leakage oracle $L_x^{k,\ell}(\cdot)$ is parametrized by a secret value x, a leakage parameter ℓ, and a security parameter k. A query to the leakage oracle consists of a function $f_i : \{0,1\}^* \to \{0,1\}^{\ell_i}$, to which the oracle answers with $f_i(x)$. We only require that the functions f_i be efficiently computable, and the total number of bits leaked is $\sum_i \ell_i \leq \ell$.

Definition 6 (Leakage-resilient hard relation.) *A relation $\mathcal{R}$ with a* PPT *sampling algorithm* KGEN$(\cdot)$ *is an ℓ-leakage resilient hard relation if:*

- *For any $(x, y) \leftarrow$ KGEN(1^k), we have $(x, y) \in \mathcal{R}$.*
- *There is a poly-time algorithm that decides if $(x, y) \in \mathcal{R}$.*

- *For all* PPT *adversaries* $\mathcal{A}^{L_x^{k,\ell}(\cdot)}$ *with access to the leakage oracle* $L_x^{k,\ell}(\cdot)$*, we have that*

$$\Pr\left[\mathcal{R}(x^*, y) = 1 \mid (x, y) \leftarrow \text{KGEN}(1^k); x^* \leftarrow \mathcal{A}^{L_x^{k,\ell}(\cdot)}(y)\right] \leq \text{negl}(k)$$

  Notice that without loss of generality, we can assume that $\mathcal{A}$ *queries* $L_x^{k,\ell}(\cdot)$ *only once with a function* f *whose output is* ℓ *bits.*

We also recall the notion of second-preimage resistant (SPR) relation, as defined in [DHLW10b].

Definition 7 (Second-preimage resistant relation.) *A relation* $\mathcal{R}$ *with a randomized* PPT *sampling algorithm* KGEN$(\cdot)$ *is second-preimage resistant if:*

- *For any* $(x, y) \leftarrow \text{KGEN}(1^k)$*, we have that* $(x, y) \in \mathcal{R}$*.*
- *There is a poly-time algorithm that decides if* $(x, y) \in \mathcal{R}$
- *For any* PPT *algorithm* $\mathcal{A}$*, we have that*

$$\Pr\left[\mathcal{R}(x^*, y) = 1 \wedge x^* \neq x \mid (x, y) \leftarrow \text{KGEN}(1^k); x^* \leftarrow \mathcal{A}(y)\right] \leq \text{negl}(k)$$

The average-case pre-image entropy of the SPR relation is defined as $\mathbf{H}_{avg}(\mathcal{R}) = \widetilde{\mathbf{H}}_\infty(X \mid Y)$ *, where the random variables* (X, Y) *are distributed according to* GEN(1^k)*, and* $\widetilde{\mathbf{H}}_\infty(X \mid Y)$ *is the average-conditional min-entropy of* X *conditioned on* Y*.*

Leakage-resilient hard relations from SPR relations. Dodis et al show that any SPR relation $\mathcal{R}$ is an ℓ-leakage-resilient hard relation with $\ell = \mathbf{H}_{avg}(\mathcal{R}) - \omega(\log k)$. Finally, we note that SPR relations are implied by the existence of one-way functions. We refer the reader to [ADW09b, DHLW10b] for more details.

2.4 Building Blocks

Here, we briefly recall some basic cryptographic primitives that we use in our main construction in Section 3.2.

Naor's Statistically Binding Commitment Scheme [Nao89]. In our main construction, we will use Naor's statistically binding bit commitment scheme based on one way functions. The commitment phase consists of two rounds: first, the verifier sends a $3k$ bit random string r, where k is the security parameter. The committer chooses a seed s for a pseudo-random generator $g : \{0,1\}^k \to \{0,1\}^{3k}$; if it wishes to commit to 0, then it sends $g(s)$, else it sends $g(s) \oplus r$. The decommitment phase simply involves the committer sending s to the receiver.

Public-coin Statistically Hiding String Commitment. We will also use a statistically hiding commitment scheme $\langle C, R\rangle$ that is public-coin with respect to the receiver. Such schemes can be constructed in constant rounds using collision-resistant hash functions [NY89, HM96, DPP97].

3 Leakage-Resilient Zero Knowledge: Interactive Case

In this section, we discuss our results on leakage-resilient zero knowledge in the interactive setting. We start by describing our model and our definition of leakage-resilient zero knowledge.

3.1 Our Definition

We consider the scenario where a malicious verifier can obtain arbitrary bounded leakage on the entire state (including the witness and the random coins) of the prover during the protocol execution. We wish to give a meaningful definition of zero knowledge interactive proofs in such a setting. To this end, we first modify the standard model for zero knowledge interactive proof system in order to incorporate leakage attacks and then proceed to give our definition. We refer the reader to Section 2.1 for the standard definitions of interactive proofs and zero knowledge.

We model the prover P and the verifier V as interactive turing machines that have the ability to flip coins during the protocol execution (such that the random coins used by a party in any round are determined only at the beginning of that round). In order to incorporate leakage attacks, we allow a malicious verifier V^* to make *adaptive* leakage queries on the state of the prover during the protocol execution. A leakage query to the prover consists of an efficiently computable function f_i (described as a circuit), to which the prover responds with $f_i(\texttt{state})$, where `state` is a variable that denotes the "current state" of the prover at any point during the protocol execution. The variable `state` is initialized to the witness of the prover. At the completion of each step of the protocol execution (that corresponds to the prover sending a protocol message to the verifier), the random coins used by the prover in that step are appended to `state`. That is, $\texttt{state} := \texttt{state} \| r_i$, where r_i denote the random coins used by the prover in that step. The verifier may make any arbitrary polynomial number of such leakage queries during the protocol execution. Unlike prior works, we do not require an a-priori bound on the total leakage obtained by the verifier in order to satisfy our definition (described below). Nevertheless, in order for our definition to be meaningful, we note that the total leakage obtained by the verifier must be smaller than the witness size.

We model the zero knowledge simulator $\mathcal{S}$ as a PPT machine that has access to a leakage oracle $L_w^{k,\lambda}(\cdot)$ that is parameterized by the honest prover's witness w, a leakage parameter λ (see below), and the security parameter k. A query to the oracle consists of an efficiently computable function $f(\cdot)$, to which the oracle answers with $f(w)$. In order to bound the total leakage available to the simulator, we consider a parameter λ and require that if the verifier obtains ℓ bits of total leakage in the real execution, then the total leakage obtained by the simulator (from the leakage oracle) must be bounded by $\lambda \cdot \ell$ bits. Finally, we require that the view output by the simulator be computationally indistinguishable from the verifier's view in the real execution. We formalize this in the definition below.

Definition 8 (Leakage-Resilient Zero Knowledge) *An interactive proof system $\langle P, V \rangle$ for a language $\mathcal{L}$ with a witness relation $\mathcal{R}$ is said to be λ-*leakage-resilient zero knowledge *if for every* PPT *machine V^* that makes any arbitrary polynomial number of leakage queries on P's state (in the manner as described above) with ℓ bits of total leakage, there exists a* PPT *algorithm $\mathcal{S}$ that obtains at most $\lambda \cdot \ell$ bits of total leakage from a leakage oracle $L_w^{k,\lambda}(\cdot)$ (as defined above) such that for every $(x, w) \in \mathcal{R}$, every $z \in \{0,1\}^*$, $\text{view}_{V^*}(x, z)$ and $\mathcal{S}^{L_w^{k,\lambda}(\cdot)}(x, z)$ are computationally indistinguishable.*

Some observations on the above definition are in order.

Leakage parameter λ. Note that when $\lambda = 0$, no leakage is available to the simulator (as is the case for the standard zero knowledge simulator). In this case, our definition guarantees the standard zero knowledge property. It is not difficult to see that it is impossible to realize such a definition. In fact, as we show in Section B, it is impossible to realize the above definition for any $\lambda < 1$, where ϵ is any constant less than 1. On the other hand, in Section 3.2, we give a positive result for $\lambda = 1 + \epsilon$, where ϵ is any positive constant. The meaningfulness of our positive result stems from the observation that when λ is close to 1, very roughly, our definition guarantees that *a malicious verifier does not learn anything from the protocol beyond the validity of the statement being proved and the leakage obtained from the prover.*

Leakage-oblivious simulation. Note that in our definition of leakage resilient zero-knowledge, (apart from the total output length) there is no restriction on the nature of leakage queries that the simulator may make to the leakage oracle. Then, since the simulator has indirect access to the honest prover's witness (via the leakage oracle), it may simply choose to leak on the witness (regardless of the leakage queries of the verifier) in order to help with the simulation of protocol messages instead of using the leakage oracle to *only* answer the leakage queries of the verifier. We stress that this issue should not affect any potential application of leakage resilient zero-knowledge that one may think of. Nonetheless, we think that this is an important issue since it relates to the meaningfulness of the definition. To this end, we note that this issue can easily handled by putting a restriction on how the simulator accesses the leakage oracle. Specifically, we can model the interaction between the simulation and the oracle such that the simulator is not allowed to look at the oracle's responses to its queries. The simulator is still allowed to look at the leakage queries of the verifier, and use them to create new queries for the oracle; however, the oracle's responses are sent directly to the verifier and the simulator does not get to seem them. We call such simulators *leakage-oblivious*. We note that the simulator that we construct for our protocol $\langle P, V \rangle$ (described in the next subsection) is leakage-oblivious.[4]

3.2 Our Protocol

We now proceed to give our construction of a leakage-resilient zero knowledge interactive proof system as per Definition 8. Very roughly speaking, our protocol can be seen as a combination of Feige-Shamir [FS89] and Goldreich-Kahan [GK96], in that we make use of equivocal commitments from the prover's side, as well as require the verifier to commit to all its challenges in advance. Note that while either of the above techniques would suffice for standard simulation, interestingly, we need to use them *together* to help the simulator handle leakage queries from a cheating verifier. We now describe our protocol in more detail.

Let P and V denote the prover and verifier respectively. Our protocol $\langle P, V \rangle$ proceeds in three stages, described as follows. In *Stage 1*, V commits to its challenge and a large random string r' using a challenge-response based PRS [PRS02] style preamble instantiated with a public-coin statistically hiding commitment scheme (see Section 2.4). In *Stage 2*, P and V engage in coin-flipping (that was initiated in Stage 1 when V committed to r') to jointly compute a random string r. Finally, in *Stage 3*, P and V run k (where k denotes the security parameter) parallel repetitions of the 3-round Blum Hamiltonicity protocol, where P uses Naor's commitment scheme (see Section 2.4) to commit to the permuted graphs in the first round. Here, for each bit commitment i, P uses a different substring r_i (of appropriate length) of r as the first message of Naor's commitment scheme. Protocol $\langle P, V \rangle$ is described in Figure 1. Intuitively, the purpose of multiple challenge response slots in Stage 1 is to allow the simulator to extract the values committed by V^* with minimal use of the leakage oracle. With the knowledge of the extracted values, the simulator can force the output of the coin-flipping to a specific distribution of its choice. This, in turn, allows the simulator to convert Naor's commitment scheme into an *equivocal* commitment scheme during simulation.

Theorem 1 *If public-coin statistically hiding commitment schemes exist, then the protocol $\langle P, V \rangle$, parameterized by ϵ, is a $(1+\epsilon)$-leakage-resilient zero knowledge proof system.*

We note that statistically hiding commitment schemes imply one-way functions, which in turn suffice for Naor's statistically binding commitment scheme used in our construction.

[4] Indeed, since we cannot rule out of obfuscation of arbitrary functionalities, we do not know how to obtain a formal proof without making the simulator leakage-oblivious.

Common Input: A k-vertex graph G.
Private Input to P: A Hamiltonian Cycle H in graph G.
Parameters: Security parameter 1^k, $n = \omega(\log(k))$, $t = 3k^4$, $\epsilon > 0$. Without loss of generality, we assume that $\frac{1}{\epsilon}$ is an integer.

Stage 1 (Commitment phase)

$V \rightleftarrows P$**:** Commit to a t-bit random string r' and $(\frac{n^2}{\epsilon})$-pairs of random shares $\left\{r'^0_{i,j}, r'^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$ (such that $r'^0_{i,j} \oplus r'^1_{i,j} = r'$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$) using a public-coin statistically hiding commitment scheme. Similarly commit to a k-bit random string ch and $(\frac{n^2}{\epsilon})$-pairs of random shares $\left\{ch^0_{i,j}, ch^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$ (such that $ch^0_{i,j} \oplus ch^1_{i,j} = ch$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$) using a public-coin statistically hiding commitment scheme.

Challenge-response slots: For every $i \in [\frac{n}{\epsilon}]$,

$P \rightarrow V$**:** Choose n-bit random strings $\alpha_i = \alpha_{i,1}, \ldots, \alpha_{i,n}$ and $\beta_i = \beta_{i,1}, \ldots, \beta_{i,n}$. Send α_i, β_i to V.

$V \rightarrow P$**:** For every $j \in [n]$, V^* decommits to $r'^{\alpha_{i,j}}_{i,j}$ and $ch^{\beta_{i,j}}_{i,j}$.

Stage 2 (Coin-flipping completion phase)

$P \rightarrow V$**:** Choose a t-bit random string r'' and send it to V.

$V \rightarrow P$**:** Decommit to r' and $r'^0_{i,j}$, $r'^1_{i,j}$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$. Let $r = r' \oplus r''$.

Stage 3 (Blum Hamiltonicity protocol)

$P \rightarrow V$**:** Let $r = r_1, \ldots, r_{k^3}$, where $|r_i| = 3k$ for every $i \in [k^3]$. For every $i \in [k]$,

- Choose a random permutation π_i and prepare an isomorphic copy of G, denoted $G_i = \pi_i(G)$.
- For every $j \in [k^2]$, commit to bit b_j in the adjacency matrix of G_i using Naor's commitment scheme with $r_{i \times j}$ as the first message.

$V \rightarrow P$**:** Decommit to ch and $ch^0_{i,j}$, $ch^1_{i,j}$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$.

$P \rightarrow V$**:** Let $ch = ch_1, \ldots, ch_k$. For each $i \in [k]$, if $ch_i = 0$, decommit to every edge in G_i and reveal the permutation π_i. Else, decommit to the edges in the Hamiltonian Cycle in G_i.

Figure 1: Protocol $\langle P, V \rangle$

3.3 Proof of Theorem 1

We start by arguing that protocol $\langle P, V \rangle$ is complete and sound. Then we argue that the protocol is $(1 + \epsilon)$-leakage-resilient zero knowledge.

Completeness. The completeness of our protocol follows directly from the completeness of Blum's Hamiltonicity protocol.

Soundness. Before we jump into the proof, we recall and build some notation related to Naor's commitment scheme (cf. Section 2.4) that we will need in our proof. This commitment scheme is statistically binding as long as the first message sent by the receiver does not come from a special set $\mathcal{B} \subset \{0,1\}^{3k}$, where $\mathcal{B}$ is the set of all strings $r = g(s_0) \oplus g(s_1)$ such that $s_0, s_1 \in \{0,1\}^k$ and $g : \{0,1\}^k \rightarrow \{0,1\}^{3k}$ is a pseudorandom generator. It follows from inspection that $\frac{|\mathcal{B}|}{2^{3k}}$ is negligible in

k. However, observe that if the first message of receiver is in fact chosen from the set $\mathcal{B}$, then Naor's commitment is no longer statistically binding and allows for equivocation.

The proof of soundness of $\langle P, V\rangle$ follows in two steps. First we argue that no cheating prover P^* can force the string r computed via coin flipping to lie in the set $\mathcal{B}$. Then, given that $r \notin \mathcal{B}$, it follows that the prover's commitments in Stage 3 are statistically binding. From this soundness follows by a standard argument in the same manner as [PRS02, Ros04]. Next, we give more details.

Stage 1 all together can be thought of as a statistically hiding commitment to r' and challenge string ch. We note that coin flipping phase (Stage 2) generates an output r which is $t = 3k^4$ bits long and is used for k^3 Naor's bit commitments. For simplicity of exposition, we restrict ourselves to the first $3k$ bits of r. These correspond to the bits that will be used as the first message of Naor's commitment scheme for the first bit commitment by the prover in Stage 3. We argue that a cheating prover can not force these $3k$ bits to lie in set $\mathcal{B}$. We can argue about the remaining bits of r in an analogous manner. Consider $r_0, r_1 \in \{0,1\}^{3k}$ with the property that there does not exist any $r^* \in \{0,1\}^{3k}$ such that $r_0 \oplus r^* \in \mathcal{B}$ and $r_1 \oplus r^* \in \mathcal{B}$. We argue that if a cheating prover P^* can force the first $3k$ bits of r to lie in $\mathcal{B}$ with non-negligible probability ε, then we can construct an adversary $\mathcal{A}$ that can distinguish between a statistically hiding commitment to r_0 and a commitment to r_1 with probability $\frac{1}{2} + \frac{\varepsilon}{2}$, thus obtaining a contradiction. Consider the adversary $\mathcal{A}$ that takes an input a statistically hiding commitment to r_b (where r_b is either r_0 or r_1) from an external challenger, and uses P^* to determine b. $\mathcal{A}$ forwards the commitment to r_b from the external challenger to P^* as its commitment to the first $3k$ bits of r'. It generates the commitments to the remaining bits of r' and the challenge string ch on its own. Let r'' be the string sent by P^*. Let r^* be the first $3k$ bits of r''. $\mathcal{A}$ outputs $b = 0$ if $r_0 \oplus r^* \in \mathcal{B}$, and $b = 1$ if $r_1 \oplus r^* \in \mathcal{B}$, and randomly guesses $b \in \{0,1\}$ otherwise.

Finally, note that the commitment to challenge ch from the verifier is statistically hiding and Naor's commitment from the prover is statistically binding. From this it follows by standard argument that if a cheating prover can convince verifier of a false theorem then we can use this prover to break the statistical hiding property of the statistically hiding commitment scheme.

Leakage-Resilient Zero Knowledge. Now we argue that the protocol $\langle P, V\rangle$ (cf. Figure 1) is $(1+\epsilon)$-leakage-resilient zero knowledge. For this we need to construct a simulator that simulates the view of every cheating verifier V^*. Our simulator has access to a leakage oracle $L_w^{k,\lambda}(\cdot)$ to help it with the leakage queries, such that if V^* queries for a total of ℓ bits of leakage then the simulator is allowed to leak $(1+\epsilon)\cdot\ell$ bits. Without loss of generality, we assume that immediately after every protocol message sent by the prover, the cheating verifier makes exactly one leakage query. However, we do not restrict the output length or the nature of these queries. In particular, these queries could change adaptively depending on the messages of the prover and the leakage itself. We stress that the above assumption has been made only to simplify exposition, and indeed, our simulator can handle arbitrary number of leakage queries as well.

We start by describing our simulator in the next section. We then discuss bounds on the total leakage required by the simulator, and finally give a proof that the view of a cheating verifier interacting with a real prover is computationally indistinguishable from the view of the verifier interacting with our simulator.

3.3.1 Description of $\mathcal{S}$

We start by giving an informal description of the simulator, and then proceed to a more formal treatment.

Informal description of $\mathcal{S}$. The purpose of Stage 1 in our protocol is to help the simulator in the extraction of r' and ch. Once a successful extraction of these values is completed, the simulator can

simulate in a "straight line" manner. Further, note that typically extraction in a "stand-alone setting" (in the absence of leakage) can be performed in expected polynomial time by rewinding multiple times in only one slot. Therefore, at first it might seem unnatural that we use $n = \omega(\log(k))$ slots. However we stress that rewinding in our case is complicated by the fact that the simulator has to respond to the leakage queries of the verifier. Whenever the simulator rewinds V^*, it might ask for a *new* leakage query (different from the one it asked on the "main" thread of execution); as a result, the total leakage required by the simulator might grow with the number of rewindings.

We deal with this issue by using the following rewinding strategy. Consider the i^{th} challenge response slot in Stage 1. We will refer to the main thread of execution with the verifier (that is output by the simulator) as the *main thread* and the execution thread created as a result of rewinding as the *look-ahead* thread. Now, consider the leakage query made by V^* immediately after the after the simulator sends a random challenge in the the i^{th} slot on the main thread. Suppose that the output length of this query is ℓ^m bits. The simulator will respond to this query using the leakage oracle $L_w^{k,\lambda}(\cdot)$ (in the manner as described later). Now, the simulator rewinds V^* *once* (in that slot) and creates a look-ahead thread, where it sends a new random challenge. The verifier may now ask for a new leakage query. Suppose that the output length of this query is ℓ^a bits. If $\ell^a \leq \ell^m$, then the simulator responds to this query using the leakage oracle $L_w^{k,\lambda}(\cdot)$ and aborts the look-ahead thread otherwise. The simulator will follow the same strategy for each slot.

Now, based on a standard "swapping argument", we can say that in each slot, in which V^* does not abort in the main thread, the simulator is able to extract r' and ch with a probability at least $1/2$. If V^* does not cause an abort in the main thread, then the simulator has n rewinding opportunities, and it will be able to extract r' and ch with overwhelming probability. This is still not good enough as the simulator might need leakage that is twice in size than what the verifier queries (whereas we want a precision of $(1+\epsilon)$). We fix this issue by having the simulator rewind in only those slots in which the leakage queries have "short output length."

We now proceed to give a formal description of the simulator. We split the simulator into three key parts, that correspond to the three stages of the protocol $\langle P, V \rangle$ (cf. Figure 1). We go over these step by step.

Description of $\mathcal{S}$ in Stage 1. Recall that in Stage 1 of the protocol, the verifier commits to a string r' and its shares shares $\left\{ {r'}^0_{i,j}, {r'}^1_{i,j} \right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$, as well as a challenge string ch and its shares $\left\{ ch^0_{i,j}, ch^1_{i,j} \right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$. Following these commitments, there are $\frac{n}{\epsilon}$ challenge-response slots between $\mathcal{S}$ and the verifier. For each $p \in \{0, \ldots, n-1\}$ consider the set of slots $\frac{p}{\epsilon}+1$ to $\frac{p+1}{\epsilon}$. Among these slots, consider the slot i in which the output length of the verifier's leakage query is the smallest (let this length be ℓ_i^m). The simulator rewinds to the point in the i^{th} slot where the challenge was sent and sends a new random challenge. At this point V^* might make a new leakage query. Let the output length of this query be ℓ_i^a bits. If $\ell_i^a \leq \ell_i^m$, then $\mathcal{S}$ uses the leakage oracle to answer the leakage query (in the manner as discussed below). Now, if V^* decommits correctly (as per the random challenge), then the simulator uses the decommitted values (obtained on the main thread and the look-ahead thread) to extract both r' and ch. It then aborts the look ahead. Further note that we consider one such slot i for each set of slots $\frac{p}{\epsilon}+1$ to $\frac{p+1}{\epsilon}$ (where $p \in \{0, \ldots, n-1\}$). If the simulator fails to extract r' and ch before the completion of Stage 1, then it aborts. This rewinding strategy is demonstrated more formally in Figure 2. Note that leakage queries have been explicitly marked by the $\twoheadleftarrow$ arrow.

Leakage Queries in Stage 1. Let $R(\cdot)$, which takes the prover's witness w as input, be a function that outputs the value of random coins of an honest prover which when used along with the prover's witness will result in the messages generated by the simulator. More specifically, the honest prover strategy

Common Input: A k-vertex graph G.
Private Input to $L(\cdot)$: A Hamiltonian Cycle H in graph G (same as real prover).
Parameters: Security parameter 1^k, $n = \omega(\log(k))$, $t = 3k^4$, $\epsilon > 0$. Without loss of generality, we assume that $\frac{1}{\epsilon}$ is an integer.

$V^* \rightleftarrows \mathcal{S}$**:** $\mathcal{S}$ acts just like a real prover and obtains the commitments to r' $\left\{r'^0_{i,j}, r'^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$, ch and $\left\{ch^0_{i,j}, ch^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$ from V^*.

$\mathcal{S} \nleftarrow V^*$**:** V^* could make multiple leakage queries in the above step. $\mathcal{S}$ uses $L_w^{k,\lambda}(\cdot)$ to answer all these leakage queries (in the manner as described in the main text). V^* could abort as well, in which case $\mathcal{S}$ aborts.

Challenge Response: For every $p \in 0, \ldots, (n-1)$,

1. For every $q \in 1, \ldots, 1/\epsilon$, do the following. Let $i = p/\epsilon + q$.
   (a) $\mathcal{S} \rightarrow V^*$: Choose n-bit random strings $\alpha_i = \alpha_{i,1}, \ldots, \alpha_{i,n}$ and $\beta_i = \beta_{i,1}, \ldots, \beta_{i,n}$. Send α_i, β_i to V.

$\mathcal{S} \nleftarrow V^*$: $\mathcal{S}$ uses $L_w^{k,\lambda}(\cdot)$ to answer the leakage queries (in the manner as described in the main text). Let the output length of the leakage query be ℓ_i^m bits.

   (b) $V^* \rightarrow \mathcal{S}$: For every $j \in [n]$, V^* decommits to $r'^{\alpha_{i,j}}_{i,j}$ and $ch^{\beta_{i,j}}_{i,j}$.

2. $\mathcal{S} \rightarrow V^*$: $\mathcal{S}$ rewinds V^* to Step 1a of slot i such that $i = \min_{j \in \{\frac{p}{\epsilon}+1, \ldots, \frac{p+1}{\epsilon}\}} \ell_j^m$. It chooses fresh n-bit random strings $\alpha'_i = \alpha'_{i,1}, \ldots, \alpha'_{i,n}$ and $\beta'_i = \beta'_{i,1}, \ldots, \beta'_{i,n}$ and sends α'_i, β'_i to V.

$\mathcal{S} \nleftarrow V^*$: Let the output length of the leakage query be ℓ_i^a bits. If $\ell_i^a \leq \ell_i^m$, then $\mathcal{S}$ uses $L_w^{k,\lambda}(\cdot)$ to answer the leakage queries. Otherwise it aborts.

3. $V^* \rightarrow \mathcal{S}$: For $j \in [n]$, V^* opens $r'^{\alpha_{i,j}}_{i,j}$ and $ch^{\beta_{i,j}}_{i,j}$ or it aborts. In either case $\mathcal{S}$ aborts the look ahead thread.

Note on leakage queries. All messages played by the simulator in Stage 1 are public coin; therefore, any leakage query from V^* can be reduced to a leakage query on only the witness (as described in the main text).

Figure 2: Rewindings in Stage 1.

with the prover's witness w and the random coins $R(w)$ will generate the exact same messages as the simulator. The function $R(\cdot)$ is initialized with the null string. Now note that in this stage, all messages played by an honest prover are public coin. Then, $R(\cdot)$ at any point in Stage 1 is just the concatenation of all the protocol messages sent by the simulator so far. Now consider a leakage query f of the adversarial verifier that takes as input the prover's witness and the random coins used by the prover so far. On receiving such a query f, the simulator creates a new query f' (that takes as input only the prover's witness w) such that $f'(w) = f(w, R(w))$. It then queries the leakage oracle with f' to obtain $f'(w)$ and returns it to the cheating verifier.

Description of $\mathcal{S}$ in Stage 2. Let r' be the random string (not including the challenge ch) extracted by the simulator in Stage 1. For every $v \in \{0, \ldots, k^3 - 1\}$, the simulator chooses $r''_v = r'_v \oplus g(s_v^0) \oplus g(s_v^1)$ (where $s_v^0, s_v^1 \in \{0,1\}^k$ are randomly chosen) and sends it to V^*. Here, r''_v and r'_v denote $3k$ bit long substrings of r'' and r' respectively, between positions $3vk + 1$ and $3(v+1)k$. Now, if V^* decommits to a value different from the extracted string r', then $\mathcal{S}$ aborts.

Leakage Queries in Stage 2. All messages played by an honest prover in Stage 2 are also public coin and just like in Stage 1, $R(\cdot)$ at any point in Stage 2 is just the concatenation of all the protocol messages sent by the simulator so far. The leakage queries of the cheating verifier are handled using $R(\cdot)$ in the same way as described earlier in Stage 1.

Description of $\mathcal{S}$ in Stage 3. Let ch denote the challenge string extracted by the simulator at the end of Stage 1. Let $ch = ch_1, \ldots, ch_k$. For each $i \in [k]$, if $ch_i = 0$, then the simulator chooses a random permutation π_i and commits to $G_i = \pi_i(G)$; otherwise, it commits to a random k-cycle graph G_i. Depending upon the verifier's challenge, it reveals the permutation π_i and decommits to the graph G_i or it decommits to the edges corresponding to the cycle in G_i. If the challenge string sent by V^* is different from ch, then $\mathcal{S}$ aborts.

Leakage Queries in Stage 3. The leakage queries in this Stage need to be handled carefully. Observe that during Stage 3, for every $i \in [k]$, an *honest prover* chooses a random permutation π_i and commits to $G_i = \pi_i(G)$. Note that during this process, an honest prover would have flipped coins to generate a random permutation π_i and commitments to G_i. To emulate honest prover behavior, $\mathcal{S}$ must be able to reconstruct the permutation π_i and the randomness used in generating commitments to G_i as a function of the witness only. Furthermore, this randomness must be consistent with what $\mathcal{S}$ later reveals (while decommitting) in Stage 3. There are two cases.

1. If $ch_i = 0$, then, as mentioned earlier, $\mathcal{S}$ chooses a random permutation π_i and commits to $G_i = \pi_i(G)$. Let R' denote the random coins used by $\mathcal{S}$ to generate the commitments. In this case $\mathcal{S}$ updates $R(\cdot)$ as $R(\cdot)\|\pi_i\|R'$.

2. The case when $ch_i = 1$ is slightly more involved. In this case, as mentioned earlier, $\mathcal{S}$ commits to a random k-cycle graph G_i. For the edges that are not part of the cycle, it commits in a way so that it can equivocate. Observe that later in the simulation $\mathcal{S}$ will actually reveal the cycle and decommit the edges on the cycle. Hence the cycle and the openings to the commitments that correspond to the edges of the cycle are fixed. Now, intuitively, by "using the witness" $\mathcal{S}$ can map the cycle in G with the cycle in G_i and obtain a permutation π_i. $\mathcal{S}$ then computes $G^* = \pi_i(G)$ and uses equivocation to explain commitments that it sent earlier as if they were for G^*. However, it must do all this in a setting where it has access to the witness only via the leakage oracle.

   More formally, consider a string ρ that consists of the following values. First, for each edge belonging to the cycle in G_i, ρ consists of the random coins that $\mathcal{S}$ used when it committed to bit 1 for that edge. Further, for each other edge (not in the cycle) or a non-edge, ρ consists of both the random coins that result in a commitment to bit 1 and the random coins that result in a commitment to bit 0. Note that the simulator can compute the random coins for both cases since it can equivocate. Now consider the function $R'(G, \rho, w)$ that works as follows. It first superimposes the cycle graph G_i onto G such that the cycle in G (determined by w) maps to the cycle in G_i. Note that this can be done in multiple ways. The function R' picks one such mapping randomly. It then obtains the permutation π_i that would lead to this mapping. Now, let G^* denote the graph such that $\pi_i(G) = G^*$. Note that G^* consists of the same k-cycle as in G_i, while the remaining structure of the graph may be different. Now, the function R' determines the random coins (from ρ) that when used to commit to (the adjacency matrix for) G^* would result in the same commitment string as the one that $\mathcal{S}$ sent earlier (when it committed to G_i). For the edges corresponding to the cycle in G^*, R' selects from ρ the (unique) random coins corresponding to the edges belonging to the cycle in G_i. Further, for each other edge (not in the cycle) or non-edge in G^*, (depending upon whether it is an edge or a non-edge) R' selects

the *appropriate* corresponding random coins from ρ (where the correspondence is determined by the mapping obtained above). Let R'' denote the concatenation of all the random coins selected from ρ in the above manner. Finally, R' outputs $\pi_i \| R''$. Now the simulator updates the function $R(\cdot)$ as $R(\cdot) \| R'(G, \rho, \cdot)$.

The leakage queries of the cheating verifier are handled using $R(\cdot)$ in the same way as described earlier in Stage 1.

3.3.2 Total leakage queries by $\mathcal{S}$

Lemma 1 *If in a protocol execution V^* makes queries with a total leakage of ℓ bits then the simulator $\mathcal{S}$ only requires $(1+\epsilon) \cdot \ell$ bits of leakage.*

Proof. This follows directly from the construction of our simulator. Consider the first $\frac{1}{\epsilon}$ slots in Stage 1 of the protocol. In the first $\frac{1}{\epsilon}$ slots, consider the slot where the verifier makes a leakage query with the smallest output length. Let f denote this leakage query. It follows from the simulator's description that the simulator rewinds once in this slot, and the verifier makes at most one leakage query on the look-ahead thread created as a result of the rewinding. Let f^* denote the leakage query made by the verifier on the look-ahead thread. Note that the output length of f^* is no more than the output length of f. Now, suppose that the total leakage obtained by the verifier during the first $1/\epsilon$ slots on the main thread is ℓ_1 bits (such that $\sum_i \ell_i = \ell$). Then, by piegonhole principle, the output length of f (and therefore, f^*) cannot be greater than $\epsilon \cdot \ell_1$. Thus the total leakage obtained by the verifier during the first $\frac{1}{\epsilon}$ slots is $(1+\epsilon) \cdot \ell_1$.

The same reasoning applies to each set of $\frac{1}{\epsilon}$ slots, and therefore, the total leakage is upper bounded by $(1+\epsilon) \cdot \ell$. □

3.3.3 Indistinguishability of the views

We now need to prove that view of V^* generated in interaction with the real prover is indistinguishable from the view generated when interacting with the simulator $\mathcal{S}$. We start by describing our hybrids.

- $\mathcal{H}_0$ This hybrid corresponds to the view of the verifier V^* in interaction with $\mathcal{S}$ when it has the witness and follows honest prover strategy. This corresponds to the real interaction. Leakage queries are answered directly based on the witness and the public coins used by the simulator.

- $\mathcal{H}_1$ This hybrid is just like in $\mathcal{H}_0$, except that the simulator $\mathcal{S}$ rewinds V^* in n challenge response slots of Stage 1 as explained in Figure 2. $\mathcal{S}$ aborts if the main thread reaches end of Stage 1 but r' and ch have not been extracted. The simulator has the witness and the leakage queries are answered in the same way as in $\mathcal{H}_0$.

- $\mathcal{H}_2$ This hybrid is just like in $\mathcal{H}_1$, except that $\mathcal{S}$ aborts if V^* opens r' and ch differently from the extracted values. Leakage queries are answered in the same way as in $\mathcal{H}_1$.

- $\mathcal{H}_3$ This hybrid is same as $\mathcal{H}_2$, except that instead of sending a random string r'' to V^*, $\mathcal{S}$ does the following. For every $v \in \{0, \ldots, k^3 - 1\}$, $\mathcal{S}$ chooses $r''_v = r'_v \oplus g(s^0_v) \oplus g(s^1_v)$ (where $s^0_v, s^1_v \in \{0,1\}^k$ are randomly chosen) and sends it to V^*. (Here, r''_v and r'_v denote $3k$ bit long substrings of r'' and r' respectively, between positions $3vk+1$ and $3(v+1)k$.) Further, the v^{th} commitment in Stage 3 is made by sending $g(s^0_v)$. It can be opened to 0 by sending s^0_v and to 1 by sending s^1_v. The simulator has the witness and the leakage queries are answered in the same way as in $\mathcal{H}_2$.

$\mathcal{H}_4$ $\mathcal{H}_4$ is different from $\mathcal{H}_3$ only in the commitments that prover makes in the Stage 3. Let $ch = ch_1, \ldots, ch_k$ be the challenge string that $\mathcal{S}$ extracted in Stage 1. For each $i \in [k]$, if $ch_i = 0$, then $\mathcal{S}$ chooses a random permutation π_i and commits to $G_i = \pi_i(G)$. Else, $\mathcal{S}$ commits to a random k-cycle graph G_i. Depending upon the verifier's challenge, it reveals the permutation π_i and decommits to the graph G_i or it decommits to the edges corresponding to the cycle in G_i. Leakage queries are handled as described in the description of the simulator. Note that simulator only needs access to a leakage oracle to answer the leakage queries. Note that $\mathcal{H}_4$ corresponds to the simulator described earlier.

Indistinguishability of $\mathcal{H}_0$ and $\mathcal{H}_1$. The only difference between hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ is that the simulator may abort in $\mathcal{H}_1$ at the end of Stage 1. Now, consider the event E that the prover in $\mathcal{H}_1$ reaches the end of the Stage 1 (Commitment Phase) but fails to extract r and ch (and thus aborts). From Lemma 2 (given below) it follows that the probability of event E is negligible and therefore the hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ are statistically close.

Indistinguishability of $\mathcal{H}_1$ and $\mathcal{H}_2$. We note that V^* can not open commitments to r' and ch differently from the extracted values, because the commitment $\langle C, R\rangle$ being used is computationally binding. If V^* opens any of the commitments in two different ways with a non-negligible probability then we can use V^* to construct an adversary that breaks the computational binding property of the used commitment scheme. Then, it follows that $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable.

Indistinguishability of $\mathcal{H}_2$ and $\mathcal{H}_3$. If an adversary can distinguish between hybrids $\mathcal{H}_2$ and $\mathcal{H}_3$ then we can use this adversary to break the security of the pseudo-random generator used in our instantiation of Naor's commitment scheme. For this consider a sequence of hybrids $\mathcal{H}_{2,0} \ldots \mathcal{H}_{2,k^3}$. In $\mathcal{H}_{2,i}$ the bits in r that correspond to the first i commitments are created in a way as specified in the hybrid $\mathcal{H}_3$, the rest are created as in $\mathcal{H}_2$. Observe that hybrid $\mathcal{H}_{2,0}$ is same as hybrid $\mathcal{H}_2$ and hybrid $\mathcal{H}_{2,k^3}$ is same as hybrid $\mathcal{H}_3$. Now we argue that if an adversary $\mathcal{D}$ can distinguish between hybrids $\mathcal{H}_{2,i}$ and $\mathcal{H}_{2,i+1}$ then we can use this adversary to construct an adversary $\mathcal{A}$ that can distinguish a random string from a pseudorandom string. The argument depends on the value being committed. $\mathcal{A}$ obtains a string a and it is supposed to guess if it is random or pseudorandom. It picks a random string s^1 and evaluates $g(s^1)$. It forces the bits of r that corresponding to the i^{th} commitment to $g(s^1) \oplus a$. And sends its commitment as $g(s^1)$ if it needs to commit to 0 and as a if it needs to commit to 1. The distinguishing advantage of $\mathcal{D}$ directly translates to the distinguishing advantage of $\mathcal{A}$.

Indistinguishability of $\mathcal{H}_3$ and $\mathcal{H}_4$. Finally, we note that hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$ are identical. The only change in $\mathcal{H}_4$ form $\mathcal{H}_3$ is that the simulator does not know the witness and hence does not know the openings of the commitments that it does not open in the protocol. But even though $\mathcal{S}$ does not know the openings and can not compute them efficiently, the openings itself come from the same distribution as $\mathcal{H}_3$. And the simulator having access to the leakage oracle can evaluate these openings and answer leakage queries correctly. It follows from the description of the simulator that it responds to the leakage queries in exactly the same manner as hybrid $\mathcal{H}_3$, therefore the view of the cheating verifier with respect to the leakage queries is identical in hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$.

Lemma 2 *Consider the event E that the simulator reaches the end of the Stage 1 but fails to extract r and ch. Then,*

$$\Pr[E] \leq \frac{1}{2^n}$$

Proof. Consider the event E_i such that:

1. In the i^{th} challenge response slot V^* responds to the challenge in the main thread after a leakage query of output length ℓ_i^m.

2. When V^* is rewound in the i^{th} challenge response slot then V^* makes a query with output length ℓ_i^a in the look ahead thread such that $\ell_i^a > \ell_i^m$ or it aborts after a leakage query of length $\ell_i^a \leq \ell_i^m$.

Lets say that the challenge sent in the main thread is c and the challenge sent in the look ahead is c' such that the event E_i happens. We ignore the case in which $c = c'$ as this happens with negligible probability. It can been seen that since both c and c' are chosen randomly, it is equally likely that challenge c' was chosen in the main thread and c was chosen in the look ahead thread. In that case, V^* would not abort in the look ahead thread and if it does not abort in the main thread then the output length of the leakage query in the look ahead would be smaller than the output length of the leakage query in the main thread. In nutshell, we have argued that for every choice of challenges which leads to event E_i there exists another choice which does not lead to event E_i. Hence, $\Pr[E_i] \leq 1/2$. This holds for every i by the same argument. Note that each E_i is an independent event and since the simulator gets to rewinds in k different slots the probability that it fails to extract in all of them is negligible.

$$\begin{aligned}\Pr[E] =& \Pr[\bigwedge_{i=1}^{k} E_i] \\ =& \bigwedge_{i=1}^{k} \Pr[E_i] \\ \leq& \frac{1}{2^n}\end{aligned}$$

□

3.4 Leakage-Resilient Zero Knowledge Proofs of Knowledge

Very informally, an interactive proof system is a *proof of knowledge* if not only does the prover convince the verifier of the validity of the statement, but it also possesses a witness for the statement. This intuition is formalized by showing the existence of an *extractor* machine, that is able to extract a witness from a prover that succeeds in convincing an honest verifier. The proof of knowledge property can be useful in several applications. In particular, in our construction of a UC secure computation protocol (see Section 5) in the "leaky token model", we will need a λ-leakage-resilient zero knowledge proof of knowledge (LR-ZKPOK) system.

We note that the protocol $\langle P, V \rangle$ described earlier is not a proof of knowledge. Very roughly, note that since the verifier challenge (to be used in Stage 3) is committed to in advance in Stage 1, the standard extractor algorithm for Blum's Hamiltonicity protocol cannot be used here. To this end, we now briefly discuss how to modify protocol $\langle P, V \rangle$ to incorporate the proof of knowledge property.

In the modified protocol, in Stage 3, the verifier simply reveals the value *ch* (without decommitting) and additionally engages in an execution of a *public-coin* zero knowledge proof of knowledge $\langle P', V' \rangle$ to prove that the revealed value *ch* is correct. Now, during the rewindings, the extractor algorithm can simply send a random challenge string and use the simulator for $\langle P', V' \rangle$ to convince the prover that the revealed value is correct.

We note that while the above modification seems to work fine for extraction purposes, we need to verify that it does not adversely affect the leakage-resilient zero knowledge property of the original protocol. Specifically, note that since a cheating verifier is allowed to make arbitrary leakage queries in

our model, we would require that protocol $\langle P', V' \rangle$ remains *sound* even when P' (played by the verifier of $\langle P, V \rangle$) can obtain arbitrary leakage information from V' (played by the prover of $\langle P, V \rangle$). To this end, we note that since $\langle P', V' \rangle$ is *public-coin*, leakage queries do not reveal any useful information to P' as long as it cannot leak on *future random coins*, which is indeed the case in our model. We note that proof of Theorem 1 given in previous subsection can be easily extended to account for these changes.

4 Leakage-Resilient NIZK

In this section, we discuss our results on leakage-resilient NIZKs. To begin with, we describe our (leakage) model and give our definition of leakage-resilient NIZKs. We refer the reader to Section 2.2 for the standard definition of non-interactive zero knowledge proof systems. Below, we will follow the notation introduced in Section 2.2.

4.1 Our Definition

We consider the scenario where a malicious verifier can obtain arbitrary leakage on the witness and the random coins used by an honest prover to generate the proof string. To model leakage attacks, we allow the cheating verifier to make adaptive leakage queries on the honest prover's witness and the random coins used to generate the proof string. A leakage query to the prover consists of an efficiently computable function f, to which the prover replies with $f(w\|r)$, where w and r denote the prover's witness and random coins respectively. It is easy to see that in the non-interactive proofs setting, a cheating verifier who is allowed multiple leakage queries enjoys no additional power than one who is allowed only one leakage query. Therefore, for simplicity of exposition, from now on, we only consider cheating verifiers who make only one leakage query. We note that our definition given below can be easily adapted to incorporate multiple leakage queries. [5]

We model the zero knowledge simulator $\mathcal{S}$ as a PPT machine that has access to a leakage oracle $L_w^k(\cdot)$ that is parameterized by the honest prover's witness w and the security parameter k. (Unlike the interactive proofs setting, here we do not consider the leakage parameter λ for simplicity of exposition.) The leakage oracle accepts queries of the form f (where $f(\cdot)$ is an efficiently computable function) and outputs $f(w)$. In order to bound the total leakage available to the simulator, we require that if the verifier obtains ℓ bits of total leakage from the honest prover, then the total leakage obtained by the simulator (from the leakage oracle) must be bounded by ℓ bits.

Definition 9 (LR-NIZK) *A non-interactive proof system (K, P, V) for a* PPT *relation $\mathcal{R}$ is said to be a* leakage-resilient NIZK *if there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that for all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k) : \mathcal{A}^{PR(\sigma,\cdot,\cdot,\cdot)}(\sigma) = 1] \stackrel{c}{\equiv} \Pr[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^k) : \mathcal{A}^{SR^{L_w^k(\cdot)}(\sigma,\tau,\cdot,\cdot,\cdot)}(\sigma) = 1],$$

where $PR(\sigma, x, w, f)$ computes $r \leftarrow \{0,1\}^{\ell_P(k)}$; $\pi \leftarrow P(\sigma, x, w; r)$; $y = f(w\|r)$ and returns (π, y), while $SR^{L_\lambda^k(\cdot)w}(\sigma, \tau, x, w, f)$ computes $r \leftarrow \{0,1\}^{\ell_{\mathcal{S}}(k)}$; $\pi \leftarrow \mathcal{S}_2(\sigma, \tau, x; r)$; $f' \leftarrow \mathcal{S}_3(\sigma, \tau, x, r, f)$; $y \leftarrow L_w^k(f')$ and returns (π, y). Here, the leakage query f' made to $L_w^k(\cdot)$ is such that its output length is no more than the output length of f. Both the oracles PR and SR output `fail` *if $(x, w) \notin \mathcal{R}$.*

[5] As in the case of leakage-resilient zero knowledge interactive proofs, we do not require an a-priori bound on the total leakage obtained by the verifier in order to satisfy our definition (described below). Nevertheless, in order for our definition to be meaningful, we note that the total leakage obtained by the verifier must be smaller than the witness size.

4.2 Our Result

We now show that every NIZK proof system with the honest prover state reconstruction property (see Section 2.2 for a formal definition) is in fact a leakage-resilient NIZK. An immediate corollary is that the Groth et al. [GOS06] NIZK proof system is a leakage-resilient NIZK proof system.

Theorem 2 *A NIZK proof system (K, P, V) for a relation $\mathcal{R}$ with honest prover state reconstruction is a leakage resilient NIZK for $\mathcal{R}$.*

Proof. Given that (K, P, V) is a NIZK proof system with honest prover state reconstruction, let $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2, \mathcal{S}'_3)$ denote a simulator for (K, P, V) as per Definition 5. Then, given such a simulator $\mathcal{S}'$, we show how to construct a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ that satisfies Definition 9.

The machine $\mathcal{S}_1$ is identical to $\mathcal{S}'_1$ in that on input 1^k, it samples a CRS string σ along with a trapdoor τ. Similarly, the machine $\mathcal{S}_2$ is identical to $\mathcal{S}'_2$ in that on input a CRS string σ, trapdoor τ, statement x and randomness ρ, it outputs a proof string π. The machine $\mathcal{S}_3$ works as follows. It takes as input a CRS string σ, trapdoor τ, statement x, randomness ρ, and a leakage query f, and outputs the description of a function f' (that only takes the witness w as input), described as follows. The function f' on input the witness w first runs the machine $\mathcal{S}'_3(\sigma, \tau, x, w, \rho)$ to obtain a random string r and then computes and outputs $f(w\|r)$. Note that f' has the CRS σ, trapdoor τ, statement x, and randomness ρ hardwired in it. Furthermore, it follows from the description of f' that the output lengths of f' and f are equal.

We now argue that the simulated view of the adversary is indistinguishable from its real view. To this end, first note that the adversary's real (resp., simulated) view only consists of the proof string π^* (resp., π) and the leakage y^* (resp, y) obtained from the honest prover (resp., simulator $\mathcal{S}$). Further, note that y^* is a function of the witness w and the honest prover's randomness (say) r^* (used to compute π^*), while y is a function of w and the honest prover's state r reconstructed by $\mathcal{S}'_3$. Then, observe that to argue the indistinguishability of adversary's views, it suffices to argue that the joint distribution of (π, w, r) is indistinguishable from the joint distribution of (π', w, r'). However, we note that this already follows from the honest prover state reconstruction property of (K, P, V). This completes the proof. □

5 UC with Leaky Tokens

Starting with the work of Goldreich and Ostrovsky on software protection [GO96], tamper-proof hardware tokens have been used for a variety of cryptographic tasks such as achieving universal composability [Kat07, CGS08, MS08, DNW09], one-time-programs [GKR08], unconditionally secure protocols [GIS+10, GIMS10], compilers for leakage-resilient computation [JV10, GR10], etc. To the best of our knowledge, all prior works using tamper-proof hardware tokens make the assumption that the tokens are completely leakage-resilient (i.e., a token does not leak any information to an adversary in possession of the token). Here, we start a new line of research to investigate whether it is possible to relax this assumption for various cryptographic tasks. In particular, in this section, we study the feasibility of doing universally composable secure computation using "leaky" tokens. More specifically, we start with the tamper-proof hardware token model of Katz [Kat07] and modify it appropriately to incorporate "bounded" leakage. Then, by making use of leakage-resilient hard relations [DHLW10b] and our leakage-resilient zero knowledge proof of knowledge, we give a construction for a universally composable multi-party computation protocol in the leaky token model.

The rest of this section is organized as follows. We first recall the hardware token model of Katz and describe our modification to incorporate leakage attacks in Section 5.1. Next, in Section 5.2, we recall the notion of "UC-puzzles" [LPV09] that is central to our positive result. Finally, we describe our positive result in Section 5.3.

5.1 Tamper-proof Hardware Setup

In the tamper-proof hardware model [Kat07], it is assumed that all parties in the system can exchange tamper-proof hardware tokens with each other. Specifically, in this model, a party can take some software code and "seal" it inside a tamper-proof hardware token; the party can then give this token to another party, who can then access the embedded software in a black-box manner. Here, the first party is referred to as the token *creator*, while the other party is referred to as the token's *user*. This setup is modeled by a "wrapper" functionality $\mathcal{G}_{wrap}$ that accepts two types of messages: the first type is used by a party P to create a hardware token (encapsulating an interactive protocol M) and to "send" this token to another party P'. $\mathcal{G}_{wrap}$ enforces that P can send at most one token to P' which is used for all their protocol interactions throughout their lifetimes (and not just for the interaction labeled by the *sid* used when the token is created). Once the token is "created" and "sent" to P', this party can interact with the token in an arbitrary black-box manner. This is formalized by allowing P' to send messages of its choice to M via the wrapper functionality $\mathcal{G}_{wrap}$. Note that each time M is invoked, fresh random coins are chosen for M. Finally, note that $\mathcal{G}_{wrap}$ prevents the token creator P from sending any messages to the token once it is "sent" to P'. The functionality $\mathcal{G}_{wrap}$ (as defined in [Kat07])) is described in Figure 3.

Functionality $\mathcal{G}_{\mathbf{wrap}}$

$\mathcal{G}_{wrap}$ is parameterized by a polynomial p and an implicit security parameter k.

Creation. Upon receiving $(\texttt{create}, sid, P, P', M, n)$ from P, where P' is another user in the system and M is an interactive Turing machine, do:

- Send $(\texttt{create}, sid, P, P')$ to P'.
- If there is no tuple of the form $(P, P', \star, \star, \star, \star)$ stored, then store $(P, P', M, n, 0, \emptyset)$.

Execution. Upon receiving $(\texttt{run}, sid, P, \mathsf{msg})$ from P', find the unique stored tuple $(P, P', M, n, i, state)$ (if no such tuple exists, then do nothing). Then, choose random $r \leftarrow \{0,1\}^{p(k)}$. Run $M(\mathsf{msg}; r; state)$ for at most p(k) steps, and let out be the response (set $\mathsf{out} = \perp$ if M does not respond in the allotted time). Send (sid, P, out) to P', and:

Case 1 ($\mathbf{i < n - 1}$): Store $(P, P', M, n, i+1, (\mathsf{msg} \| r \| state))$ and erase $(P, P', M, n, i, state)$.

Case 2 ($\mathbf{i = n - 1}$): Store $(P, P', M, n, 0; \emptyset)$ and erase $(P, P', M, n, i, state)$.

Figure 3: The wrapper functionality [Kat07].

The Leaky Token Model. We wish to weaken the assumption about the "tamper-proofness" of the hardware tokens by allowing "bounded" leakage of the secret state of a token to its user. To this end, we consider a modified wrapper functionality $\mathcal{G}^{\ell}_{wrap}$ parametrized by a leakage-parameter ℓ that defines the "total" leakage available to a token user over all the executions of the token. More concretely, the new wrapper functionality $\mathcal{G}^{\ell}_{wrap}$ is defined in the same manner as $\mathcal{G}_{wrap}$, except that $\mathcal{G}^{\ell}_{wrap}$ accepts special $\texttt{leak}$ queries (from the token user) that consist of a length-decreasing function $f_i : \{0,1\}^* \rightarrow \{0,1\}^{\ell_i}$ (described as a circuit), to which the functionality answers with $f(M, state)$, where M denotes the code of the interactive Turing machine encapsulated in the token and *state* denotes the current state of M consisting of all the protocol messages received from the user and the random coins used so far by M in the current protocol execution. The token user can make any

arbitrary polynomial number of such leakage queries over multiple protocol executions with M; we only require that the functions f_i be efficiently computable, and the total number of bits leaked (over all executions) is $\sum_i \ell_i = \ell$. The functionality $\mathcal{G}^{\ell}_{wrap}$ is described in Figure 4.

We stress that by allowing leakage on M, we essentially allow the token user to obtain leakage on any secret values hardwired into M. We also stress that it is important for our purposes that the wrapper functionality $\mathcal{G}^{\ell}_{wrap}$ flip fresh random coins (to be used by M) during each round of a protocol execution between M and the token user. (In contrast, in the original model of Katz [Kat07], the wrapper functionality may choose and a fix a random tape for M before the start of a protocol execution.[6])

Functionality $\mathcal{G}^{\ell}_{\mathbf{wrap}}$

$\mathcal{G}^{\ell}_{wrap}$ is parameterized by a polynomial p, a leakage parameter ℓ and an implicit security parameter k.

Creation. Upon receiving $(\mathtt{create}, sid, P, P', M, n)$ from P, where P' is another user in the system and M is an interactive Turing machine, do:

- Send $(\mathtt{create}, sid, P, P')$ to P'.
- If there is no tuple of the form $(P, P', \star, \star, \star, \star, \star)$ stored, then store $(P, P', M, n, 0, \emptyset, \ell)$.

Execution. Upon receiving $(\mathtt{run}, sid, P, \mathsf{msg})$ from P', find the unique stored tuple $(P, P', M, n, i, state, \delta)$ (if no such tuple exists, then do nothing). Then, choose random $r \leftarrow \{0,1\}^{p(k)}$. Run $M(\mathsf{msg}; r; state)$ for at most p(k) steps, and let out be the response (set $\mathsf{out} = \perp$ if M does not respond in the allotted time). Send (sid, P, out) to P', and:

Case 1 ($\mathbf{i < n-1}$): Store $(P, P', M, n, i+1, (\mathsf{msg} \| r \| state), \delta)$ and erase $(P, P', M, n, i, state, \delta)$.

Case 2 ($\mathbf{i = n-1}$): Store $(P, P', M, n, 0; \emptyset, \delta)$ and erase $(P, P', M, n, i, state, \delta)$.

Leakage. Upon receiving $(\mathtt{leak}, sid, P, f)$, find the unique stored tuple $(P, P', M, n, i, state, \delta)$ (if no such tuple exists, then do nothing). If $|f| > \delta$, then do nothing. Otherwise, do:

- Compute $z = f(M \| state)$ and send (sid, P, z) to P'.
- Store $(P, P', M, n, i, state, \delta - |f|)$ and erase $(P, P', M, n, i, state, \delta)$.

Figure 4: The new wrapper functionality $\mathcal{G}^{\ell}_{wrap}$ that allows ℓ bits of leakage.

5.2 UC-Security via UC-Puzzles

In order to obtain our positive result, we build on the recent work of Lin, Pass and Venkitasubramaniam [LPV09] which puts forward a unified framework for designing UC secure protocols from known setup assumptions like CRS [CF01, CLOS02], tamper-proof hardware tokens [Kat07], key registration [BCNP04], etc. As observed by Lin et al., it is implicit from prior works (see e.g. [CLOS02]) that the task of constructing UC-secure protocols for any well-formed functionality [CLOS02] reduces to the task of constructing a "concurrent simulation-sound" zero knowledge protocol (ssZK) with "UC simulation" property[7].[8] Very informally, these properties can be described as follows (the text is taken

[6] Note that in this case, if a token user were allowed leakage queries, then it would be able to leak on the entire random tape of M at the start of the protocol execution. We do not consider such a model in this paper.

[7] Formally, this can be modeled as implementing a specific "zero knowledge proof of membership" functionality.

[8] Intuitively, this is because given a functionality f, we can start with a semi-honest secure computation protocol Π for f, and then "compile" Π with an ssZK protocol to obtain a UC-secure protocol against active adversaries.

almost verbatim from [LPV09]):

UC simulation: For every PPT adversary $\mathcal{A}$ receiving "honest" proofs of statements x using witness w, where (x, w) are chosen by the environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ (that only gets statements x as input) such that no $\mathcal{Z}$ can distinguish (except with negligible probability) whether it is interacting with $\mathcal{A}$ or $\mathcal{S}$.

Concurrent simulation-soundness: An adversary who receives an unbounded number of concurrent *simulated* proofs, of statements chosen by $\mathcal{Z}$, cannot prove any false statements (except with negligible probability).

Lin et al. consider a modular approach towards constructing an ssZK protocol. They observe that a general technique for realizing the "UC simulation" property is to have the simulator obtain a "trapdoor" which is hard to compute for the adversary. This is formalized in the form of (two party) "UC-puzzle" protocols that enable the simulator to obtain such a trapdoor string (but prevent the adversary from doing so), as described below.

UC-puzzle. Let $\mathcal{G}$ denote a setup functionality. A UC-puzzle is a pair $(\langle S, R\rangle, \mathcal{R})$, where $\langle S, R\rangle$ is a protocol between two parties—a sender S, and a receiver R—in the $\mathcal{G}$-hybrid model and $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is an associated PPT computable relation. A UC-puzzle must satisfy the following two properties.

Soundness No PPT adversarial receiver R^* after an execution with an honest sender S can find (except with negligible probability) a trapdoor $\sigma \in \mathcal{R}(\texttt{trans})$, where trans is the transcript of the puzzle execution.

Statistical Simulatability Let $\mathcal{A}$ be a real world adversary (in an environment $\mathcal{Z}$) that participates as a sender in multiple concurrent executions of a UC-puzzle. Then, for every such $\mathcal{A}$, there exists a simulator $\mathcal{S}$ interacting only with $\mathcal{Z}$ such that no (possibly unbounded) $\mathcal{Z}$ can distinguish between an execution with $\mathcal{A}$ from an execution with $\mathcal{S}$, except with negligible probability. Further, for every completed puzzle execution, except with negligible probability, $\mathcal{S}$ outputs a trapdoor $\sigma \in \mathcal{R}(\texttt{trans})$, where trans is the transcript of that puzzle execution.

Now that we have a means for"UC simulation", in order to achieve "simulation-soundness", Lin et al define and construct a strongly non-malleable witness indistinguishable (SNMWI) argument of knowledge from one way functions. Lin et al. then give a construction for an ssZK protocol from a UC-puzzle and an SNMWI protocol.

We note that following the work of [LPV09], the task of constructing UC secure protocols from any setup assumption reduces to the task of constructing a UC-puzzle (in the hybrid model of the corresponding setup). We obtain our positive result by following the same route, i.e., constructing a UC-puzzle in the leaky token model. We in fact construct a "family of UC-puzzles" in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model. More details follow in the next subsection.

5.3 Our Protocol

Recall that in the hardware token model, each pair of parties in the system exchange hardware tokens with each other. Now consider a system with m parties $P_1, \ldots, P_m$. For each pair of parties (P_i, P_j), we will construct two different UC-puzzles, (a) one where P_i (resp., P_j) acts as the puzzle sender (resp., receiver) and (b) the other where the roles of P_i and P_j are reversed. This gives us a family of m^2 UC-puzzles.

Now, given such a family of UC-puzzles, we can construct a family of ssZK protocols where the protocols in the family are concurrent simulation-sound with respect to each other. Specifically, for each pair of parties (P_i, P_j), we can construct two different ssZK protocols, (a) one where P_i (resp., P_j) acts as the prover (resp., verifier), and (b) the other, where the roles of P_i and P_j are reversed. Finally, in order to construct a UC-secure protocol for any well-formed functionality f, we can start with a semi-honest protocol Π for f, and then "compile" Π with the above family of ssZK protocols in the following manner. Whenever a party P_i sends a protocol message to P_j, it proves that it has "behaved honestly so far in the protocol" by running an execution of the "appropriate" ssZK protocol (i.e., where P_i and P_j play the roles of the prover and verifier respectively) from the above family.

We now give the construction of a family of UC-puzzles in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model. Specifically, we construct a family of protocol and relation pairs $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$, where $i, j \in [m]$. Here the choice of notation is to highlight that party P_i (resp., P_j) plays the role of the sender (resp., receiver) in protocol $\langle S_{ij}, R_{ij}\rangle$. We will then prove that each pair $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$ is a UC-puzzle in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model.

Our construction of a UC-puzzle in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model is very similar to that of Lin et al [LPV09] (in the $\mathcal{G}_{wrap}$-hybrid model). Specifically, instead of using a standard witness-hiding proof of knowledge protocol, we use a λ-leakage-resilient zero knowledge proof of knowledge (LR-ZKPOK) protocol (see Section 3.2). Further, instead of using an ordinary one-way function, we use an ℓ'-leakage-resilient hard relation, as defined by Dodis, Haralambiev, Lopez-Alt, Wichs [DHLW10b], for $\ell' = \lambda \cdot \ell$. We refer the reader to Section 2.3 for a discussion on leakage-resilient hard relations. We now proceed to describe our construction.

Description of $\langle \mathbf{S_{ij}}, \mathbf{R_{ij}}\rangle$. The interactive Turing machine S_{ij}, when invoked with the inputs the identity of the sender P_i, the identity of the receiver P_j and the session id sid, proceeds as follows. It first checks whether this is the first time interacting with party P_j. If so, it first samples a pair (x, y) from an ℓ'-leakage resilient hard relation $\mathcal{R}_{\ell'}$ and then "creates" and "gives" P_j a token, which encapsulates the interactive Turing machine M that gives a λ-LR-ZKPOK of the statement that there exists an x such that $(x, y) \in \mathcal{R}_{\ell'}$. In order to "give" the token to P_j, S_{ij} sends the message $(\texttt{create}, sid, P_i, P_j, M, n)$ to $\mathcal{G}^{\ell}_{wrap}$, where n denotes the round-complexity of our λ-LR-ZKPOK protocol. To actually challenge P_j, S_{ij} simply sends y as the puzzle to the receiver.

The interactive Turing machine R_{ij}, on receiving y from S_{ij}, engages in an execution of our λ-LR-ZKPOK protocol with M (via $\mathcal{G}^{\ell}_{wrap}$) where M proves that there exists an x such that $(x, y) \in \mathcal{R}_{\ell'}$. More specifically, in order to send a protocol message msg to M, R_{ij} sends $(\texttt{run}, sid, P_i, \mathsf{msg})$ to $\mathcal{G}^{\ell}_{wrap}$. An adversarial receiver R_{ij} may additionally send leakage queries $(\texttt{leak}, sid, P, f)$ to $\mathcal{G}^{\ell}_{wrap}$, who responds with $f(M\|r)$ (where r denotes the random coins used by M "so far") as long as the total leakage (over all queries) is bounded by ℓ.

Description of $\mathcal{R}_{\mathbf{ij}}$. The puzzle relation $\mathcal{R}_{ij}$ is simply $\{(x, y)|(x, y) \in \mathcal{R}_{\ell'}\}$.

This completes the description of $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$. We now prove that $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$ is a UC-puzzle in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model. To this end, we first argue that it satisfies the Soundness property.

$(\langle \mathbf{S_{ij}}, \mathbf{R_{ij}}\rangle, \mathcal{R}_{\mathbf{ij}})$ satisfies Soundness. The Soundness property follows from the following hybrid argument:

$\mathcal{H}_{\mathbf{0}}$: This hybrid corresponds to the real execution between S_{ij} and R^*_{ij} as described above. $\mathcal{G}^{\ell}_{wrap}$ answers any leakage query from R^*_{ij} as long as the total leakage is bounded by ℓ. Let p_0 denote the probability that R^*_{ij} outputs a trapdoor $x \in \mathcal{R}(y)$ in this experiment.

$\mathcal{H}_1$: This hybrid is the same as $\mathcal{H}_0$, except that we replace the honest execution of the λ-LR-ZKPOK between the token and R^*_{ij} (via $\mathcal{G}^{\ell}_{wrap}$) with a simulated execution. Specifically, we run the simulator for our LR-ZKPOK protocol that provides a simulated proof[9] to R^*_{ij}. The leakage queries made by R^*_{ij} are answered by the simulator in the following manner. On receiving a leakage query f from R^*_{ij}, the simulator prepares a query f' to the leakage oracle in the same manner as described in Section 3.2), except for the following change. The function f' now has the code of the honest prover algorithm for our λ-LR-ZKPOK hardwired in it; f' internally computes the machine code M (using the above information) in order to compute leakage on M. Here, the leakage oracle is implemented by the puzzle sender. Note that by definition (of λ-leakage resilient zero knowledge), the simulator (and therefore in turn, R^*_{ij}) obtains at most $\lambda \cdot \ell$ bits of leakage. Let p_1 denote the probability that R^*_{ij} outputs a trapdoor $x \in \mathcal{R}(y)$ (where y is the puzzle) in $\mathcal{H}_1$.

Now, note that it follows from the λ-leakage resilient zero knowledge property of our LR-ZKPOK that the views of R^*_{ij} in $\mathcal{H}_0$ and $\mathcal{H}_1$ are computationally indistinguishable. Therefore, we have that $|p_1 - p_0| \leq \text{negl}(k)$ (where k is the security parameter).

$\mathcal{H}_2$: This hybrid is the same as $\mathcal{H}_1$, except that the puzzle y is taken from an external party who samples $(x, y) \in \mathcal{R}_{\ell'}$. The leakage queries from the simulator are forwarded to the external party and the responses are sent back to the simulator. Let p_2 denote the probability that R^* outputs a trapdoor $x \in \mathcal{R}(y)$ in $\mathcal{H}_2$.

Note that the views of R^*_{ij} in $\mathcal{H}_1$ and $\mathcal{H}_2$ are identical. Therefore, we have that $p_1 = p_2$. Now, observe that in $\mathcal{H}_2$, R^*_{ij} obtains no information on x apart from $\lambda \cdot \ell$ bits of leakage. Then, since $\mathcal{R}_{\ell'}$ is an ℓ'-leakage resilient hard relation (where $\ell' = \lambda \cdot \ell$), it follows that p_2 must be negligible (in the security parameter). Finally, since $|p_2 - p_0| \leq \text{negl}(k)$, we have that $p_0 \leq \text{negl}(k)$.

We now argue that $(\langle S_{ij}, R_{ij} \rangle, \mathcal{R}_{ij})$ satisfies the Statistical Simulatability property.

$(\langle \mathbf{S_{ij}}, \mathbf{R_{ij}} \rangle, \mathcal{R}_{\mathbf{ij}})$ satisfies Statistical Simulation. The proof for Statistical Simulatability property follows exactly as in [LPV09]. We recall the argument here for completeness. The text below is taken almost verbatim from [LPV09].

To simulate a concurrent puzzle execution with an adversarial sender $\mathcal{A}$ and the environment $\mathcal{Z}$, $\mathcal{S}$ internally emulates each execution with $\mathcal{A}$ and acts as the wrapper functionality $\mathcal{G}^{\ell}_{wrap}$ for $\mathcal{A}$. Whenever $\mathcal{A}$ sends a message $(\texttt{create}, sid, P_i, P_j, M^*)$ to $\mathcal{G}^{\ell}_{wrap}$, $\mathcal{S}$ obtains the message. Later, to extract the trapdoor of a puzzle y challenged by $\mathcal{A}$ (controlling P_i) to P_j, $\mathcal{S}$ simply rewinds M^* in the LR-ZKPOK protocol to extract the witness. Note that since M^* cannot receive messages from other parties except P_j, it would never expect any new messages from parties other than P_j during rewindings. Therefore, the extraction can be finished in isolation without intervening the adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Hence we achieve perfect simulation.

Leakage Parameter ℓ. As discussed in Section 2.3, assuming one-way functions, it is possible to construct ℓ-leakage-resilient hard relations in the bounded leakage model for optimal value of ℓ, namely, $\ell = (1 - o(1))\xi$, where ξ is the length of the secret (i.e., the witness for an instance of the hard relation).

[9]Note that simulation of our LR-ZKPOK involves rewinding of the adversary which is not allowed in the UC framework. However, we stress that the rewinding is performed here only for a "soundness" argument, which can be done outside the UC framework. Elaborating further, we note that the ZK proof being given is independent of everything else in the system (except, of course, the instance of the hard relation). Therefore, we can think of the proof in isolation of the rest of the system. Now, in this setting the adversary (or, more generally the whole environment) can be used to break the zero knowledge property of our protocol in the stand-alone setting. We note that this idea has been used in several previous works, see [BS05, CGS08].

(See Section 2.3 for more details.) Combining this with our result on $(1+\epsilon)$-LR-ZKPOK (where ϵ is a positive constant) in section 3.2, we have that $\ell = \frac{(1-o(1))\xi}{1+\epsilon}$.

Family of ssZK protocols. We note that given the family of UC-puzzles $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$, the construction of a family of ssZK protocols easily follows from the techniques as described in [LPV09]. We refer the reader to [LPV09] for more details.

6 Fully Leakage-Resilient Signatures

In this section, we give a generic constructions of fully leakage-resilient (FLR) signature schemes by building on our notion of leakage-resilient NIZKs.

In order to discuss our approach, we first briefly recall the leakage-resilient signature scheme in [DHLW10b] (which in turn is based on the construction of [KV09]). Dodis et al. gave a generic construction of a leakage-resilient signature scheme in the bounded-leakage model from a leakage-resilient hard relation, and a tag-based *true simulation-extractable* (tSE) NIZK argument system. Very roughly, a tSE-NIZK system guarantees the existence of an extractor algorithm that can extract the witness for a NIZK proof output by an adversary that has oracle access to simulated proofs of true statements under tags of his choice (the tag used in the proof output by the adversary must be different from the tags used in the simulated proofs). We note that the approach of Dodis et al. is quite general, in that if we use a hard relation that is secure in the continual-leakage (CTL) model (as opposed to only the bounded-leakage model), the resultant signature scheme is also secure in the CTL model. Indeed, this is the approach followed in [DHLW10a].

In order to construct FLR signatures (that allow leakage on the entire state as opposed to only the secret key), we extend our notion of leakage-resilient NIZK to incorporate true simulation-extractability. Then, given a *true simulation-extractable leakage-resilient* (tSE-LR) NIZK argument system, we note that the construction of [DHLW10b] (resp., [DHLW10a]) can be easily modified to obtain FLR signatures in the bounded-leakage model (resp., CTL model). Finally, we note that a tSE-LR-NIZK argument system is implicit from the UC-secure NIZK of [GOS06].

The rest of this section is organized as follows. We first define the notion of true simulation-extractable leakage-resilient NIZK and give a construction for the same in Section 6.1. Next, we present our construction of an FLR signature scheme in the bounded-leakage model in Section 6.2. Finally, in Section 6.3, we briefly discuss fully leakage-resilient signatures in the CTL model.

6.1 True Simulation-Extractable Leakage-Resilient NIZK

In this section we define tag-based tSE-LR-NIZK system and give a construction for the same. Our definition can be seen as an extension of the notion of tSE-NIZKs, as defined in [DHLW10b]. Very roughly, tSE-LR-NIZK extends the notion of tSE-NIZK by allowing the adversary to obtain (in addition to simulated proofs) leakage on the witness and randomness used to generate the simulated proofs. We note that our original definition of LR-NIZK (c.f. Definition 9) does not include tags, but we stress that it can be easily extended to do so.

Definition 10 (True simulation-extractability) *Let (K, P, V) be a leakage-resilient NIZK system for a relation $\mathcal{R}$ with a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ and a leakage oracle $L_w^k(\cdot)$. We say that (K, P, V) is* true simulation-extractable *with tags if there exists a* PPT *extractor algorithm $\mathcal{E}$ such that for all adversaries $\mathcal{A}$, we have* $\Pr[\mathcal{A} \text{ wins}] \leq \text{negl}(k)$ *in the following experiment:*

1. $(\sigma, \tau) \leftarrow \mathcal{S}_1(1^k)$.

2. $(x^*, \mathtt{tag}^*, \pi^*) \leftarrow \mathcal{A}^{SR^{L^k_w(\cdot)}(\sigma,\tau,\cdot,\cdot,\cdot,\cdot)}$, *where* $SR^{L(\cdot)}(\sigma, \tau, x, w, \mathtt{tag}, f)$ *computes* $r \leftarrow \{0,1\}^{\ell_S(k)}$; $\pi \leftarrow \mathcal{S}_2(\sigma, \tau, x, \mathtt{tag}; r)$; $f' \leftarrow \mathcal{S}_3(\sigma, \tau, x, r, f)$; $y \leftarrow L^k_w(f')$ *and returns* (π, y) *(or* `fail` *if* $x \notin \mathcal{L}$*). Note that* $\mathcal{A}$ *can query* $SR^{L(\cdot)}$ *multiple times in an adaptive manner.*

3. $w^* \leftarrow \mathcal{E}(\sigma, \tau, x^*, \mathtt{tag}^*, \pi^*)$.

4. $\mathcal{A}$ *wins if: (a) the pair* $(x^*, \mathtt{tag}^*)$ *was not part of a simulator query, (b)* $V(\sigma, x^*, \mathtt{tag}^*, \pi^*) = 1$*, and (c)* $\mathcal{R}(x^*, w^*) = 0$.

Our Construction. A tag based tSE-LR-NIZK argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ follows directly from the UC-secure NIZK constructed by Groth, Ostrovsky and Sahai [GOS06]. In fact it is relatively easier to construct tSE-LR-NIZK (as opposed to obtaining UC-security). For the sake of completeness, we give the complete construction and proof in Appendix A. A large part of the construction and the proof has been taken verbatim from [GOS06].

6.2 Fully Leakage-Resilient Signatures in the Bounded Leakage Model

We first recall the definition of FLR signatures in the bounded-leakage model from [BSW11]. Some of the text below is taken verbatim from [BSW11]. Very roughly, we say that a signature scheme is fully leakage-resilient in the bounded-leakage model if it is existentially unforgeable against any PPT adversary that can obtain polynomially many signatures over messages of her choice, as well as bounded leakage information on the secret key and the randomness used by the signing algorithm throughout the lifetime of the system.[10] We define a variable `state` that is initialized to the secret key. On each signature query from the adversary, the random coins used by the signing algorithm are appended to `state`. The adversary can leak any PPT information on `state` as long as the total amount is bounded by the leakage parameter ℓ.

Definition 11 (FLR security – bounded leakage) *A signature scheme (*KeyGen, Sign, Verify*) is* ℓ*-fully-leakage-resilient in the bounded leakage model if for all* PPT *adversaries* $\mathcal{A}$*, we have that* $\Pr[\mathcal{A} \text{ wins}] \leq \text{negl}(k)$ *in the following experiment:*

1. *Compute* $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k, \ell)$*, and set* `state` $= sk$*. Give* pk *to the adversary.*

2. *Run the adversary* $\mathcal{A}$ *on input tuple* $(1^k, pk, \ell)$*. The adversary may make adaptive queries to the signing oracle and the leakage oracle, defined as follows:*

   Signing queries: *On receiving a query* m_i*, the signing oracle samples* $r_i \leftarrow \{0,1\}^*$*, and computes* $\Phi_i \leftarrow \mathsf{Sign}_{sk}(m_i; r_i)$*. It updates* `state` $:=$ `state`$\|r_i$ *and outputs* Φ_i.

   Leakage queries: *On receiving as input the description of a polynomial-time computable function* $f_j \colon \{0,1\}^* \to \{0,1\}^{\ell_j}$*, the leakage oracle outputs* $f(\mathtt{state})$.

3. *At some point,* $\mathcal{A}$ *stops and outputs* (m^*, Φ^*).

4. $\mathcal{A}$ *wins in the experiment iff:*

   - $\mathsf{Verify}_{pk}(m^*, \Phi^*) = 1$*, and*
   - m^* *was not queried to the signing oracle, and*
   - $\sum_j \ell_j \leq \ell$.

[10] We note that in the original definition of [BSW11], the adversary can obtain leakage even on the randomness used in the key-generation algorithm. In our main discussion, for the sake of simplicity, we do not consider this case. We stress, however, that our construction satisfies the original definition of [BSW11], as discussed later in the section.

Our Construction. We now give a generic construction of fully leakage-resilient signatures based on leakage-resilient hard relations and tSE-LR-NIZK arguments. Let $\mathcal{R}_\ell$ be an ℓ-leakage-resilient hard relation with a PPT sampling algorithm $\text{KGEN}(\cdot)$. Let (K, P, V) be a tag-based tSE-LR-NIZK argument system for a relation $\mathcal{R}$. The signature scheme (KeyGen, Sign, Verify) is described as follows.

- $\mathsf{KeyGen}(1^k, \ell)$: Sample $(x, y) \leftarrow \text{KGEN}(1^k)$, $\sigma \leftarrow K(1^k)$. Output $sk = x$ and $pk = (\sigma, y)$.
- $\mathsf{Sign}_{sk}(m)$: Output $\Phi = \pi$, where $\pi \leftarrow P(\sigma, y, m, x)$. (Here m is the tag in the argument.)
- $\mathsf{Verify}_{pk}(m, \Phi)$: Output $V(\sigma, y, m, \Phi)$.

Theorem 3 *If $\mathcal{R}_\ell$ is an ℓ-leakage-resilient hard relation and (K, P, V) is a tag-based true simulation-extractable leakage-resilient NIZK argument system, then (*KeyGen*,* Sign*,* Verify*) is an ℓ-fully-leakage-resilient signature scheme in the bounded-leakage model.*

Proof. Consider the following series of experiments:

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the fully-leakage-resilience experiment as described in Definition 11. Let p_0 denote the probability that $\mathcal{A}$ outputs a successful forgery in this experiment.

Hybrid $\mathcal{H}_1$. This hybrid is the same as $\mathcal{H}_0$, except for the following changes. First, during the key generation process, instead of sampling a CRS honestly, we now run the simulator of the NIZK system to generate the CRS. Further, on receiving a query m_i from $\mathcal{A}$, instead of giving an honestly generated NIZK argument to $\mathcal{A}$, the signing oracle works as follows. It runs the simulator for our NIZK system with a leakage query f_i and obtains a simulated argument π_i and the description of a function f_i'. Here, the function f_i is such that it takes as input the witness and random coins of the NIZK prover algorithm (simulator in this case) and simply outputs all the random coins. Further, f_i' is the function output by the simulator that takes as input only the witness and produces the same output as f_i (c.f. Definition 9). The signing oracle outputs $\Phi_i = \pi_i$ and gives f_i' (as private input) to the leakage oracle.

The leakage oracle on receiving a leakage query f_j from $\mathcal{A}$ works as follows. Let $f_1', \ldots, f_i'$ denote the list of functions that the leakage oracle has received from the signing oracle so far. Then, the leakage oracle first prepares a function f_j^* that takes as input only the secret key sk (which is the witness for each proof generated by the simulator above), described as follows. The function f_j^* on input the secret key sk first computes $r_1 \leftarrow f_1'(sk), \ldots, r_i \leftarrow f_i'(sk)$ to generate $\mathtt{state} = sk \| r_1 \| \ldots \| r_i$ and then outputs $f_j(\mathtt{state})$.

Let p_1 denote the probability that $\mathcal{A}$ outputs a successful forgery in this experiment. Now, it follows from the leakage-resilient zero knowledge property of (K, P, V) that the views of $\mathcal{A}$ in $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable. Then, we have that $|p_1 - p_0| \leq \text{negl}(k)$.

Hybrid $\mathcal{H}_2$. This hybrid is the same as $\mathcal{H}_1$, except that the public key component y is now taken from an external party P who samples a pair $(x, y) \leftarrow \text{KGEN}(1^k)$ such that $(x, y) \in \mathcal{R}_\ell$. Further, instead of computing the response a leakage query f_j on its own, the leakage oracle now prepares a query f_j^* (as in the previous hybrid) and forwards it to the external party P. The response from P is sent back to $\mathcal{A}$.

Let p_2 denote the probability that $\mathcal{A}$ outputs a successful forgery in this experiment. Now, note that the views of $\mathcal{A}$ in $\mathcal{H}_1$ and $\mathcal{H}_2$ are identical. Then, we have that $p_2 = p_1$.

Now, let (m^*, Φ^*) denote the forgery output by $\mathcal{A}$. We now run the extractor for the simulation-extractable leakage-resilient on input the CRS, the CRS trapdoor, $\mathtt{tag}^* = m^*$, and $\pi^* = \Phi^*$ to obtain

a witness $x^* = w^*$. It follows from the simulation-extractability of our NIZK argument system that x^* is such that $(x^*, y) \in \mathcal{R}_\ell$, except with negligible probability. That is, we have obtained a pre-image of y with probability $p = p_2 - \text{negl}(k)$. Then, it follows from the ℓ-leakage resilience of $\mathcal{R}_\ell$ that $p \leq \text{negl}(k)$. Combining this with above, we have that $p_0 \leq \text{negl}(k)$. This concludes the proof.

Leakage parameter ℓ. As discussed in Section 2.3, assuming one-way functions, it is possible to construct ℓ-leakage-resilient hard relations in the bounded leakage model for optimal value of ℓ, namely, $\ell = (1-o(1))\xi$, where ξ is the length of the secret (i.e., the witness for an instance of the hard relation). (We refer the reader to Section 2.3 for more details.) Then, instantiating our signature scheme with such a hard relation, we have that $\ell = (1 - o(1))\xi$, where ξ is the length of the secret key.

Leakage during Key Generation. We note that the signature scheme described above can in fact tolerate leakage during the key generation algorithm (thus satisfying the original definition of Boyle et al [BSW11]) if it is possible to sample CRS for the tSE-LR-NIZK argument system in an *oblivious* manner (i.e., without first computing a trapdoor string). Note that this is possible if the CRS is a common *random* string. As we show in Section A, our construction of tSE-LR-NIZK argument system indeed satisfies this property.

6.3 Fully Leakage-Resilient Signatures in the Continual Leakage Model

In Section 6.2, we considered FLR signature schemes in the bounded-leakage model, where the adversary is allowed to obtain only some bounded leakage on the secret state during the entire lifetime of the system. A more realistic model is the *continual-leakage model* (CTL), first studied by Dodis et al. [DHLW10a] and Brakerski et al [BKKV10]. We briefly recall the CTL model in the context of FLR signature schemes [BSW11, MTVY11]. Very roughly, in this model, the adversary is allowed to leak continuously from the secret state, with no bound on the total leakage obtained during the lifetime of the system. However, there are two restrictions: First, it is assumed that the a user can "refresh" (or update) the secret key regularly, and that the total leakage between two successive updates is bounded. Second, there is no leakage during the update process.[11] As in the bounded-leakage model, a variable `state` is considered that is initialized to the secret key, and is constantly updated with the randomness used by the signing algorithm. However, at the end of an update, `state` is set to the updated secret key (such that no leakage is possible on the old secret state). We refer the reader to [BSW11] and [MTVY11] for a detailed definition of a fully leakage-resilient signature scheme in the continual leakage model.

We now briefly discuss how to extend our construction of FLR signature scheme from Section 6.2 to the CTL model. We note that if we substitute the leakage-resilient hard relation in our previous construction with a *continual leakage-resilient hard relation* [DHLW10a], we immediately obtain a FLR signature scheme in the CTL model. An alternative way of looking at this is as follows. If we substitute the tSE-NIZK used in the construction of a (standard) leakage-resilient signature scheme in the CTL model in [DHLW10a] with our true simulation-extractable leakage-resilient NIZK, we immediately obtain an FLR signature scheme in the CTL model. The construction and proof details easily follow from Section 6.2 and [DHLW10a] and are therefore omitted.

[11] As observed in [BKKV10], and by Waters (noted in [DHLW10a]), there is general technique that can be used to tolerate up to logarithmic bits of leakage during the update process. More recently, Lewko et al [LLW11] give a construction for FLR signature scheme and an encryption scheme that tolerates constant fraction of leakage during the update process. We note that if we use the key pairs of the encryption scheme of [LLW11] as a hard relation, then our construction of FLR signatures will inherit the leakage bounds of their encryption scheme.

6.4 Security in the Noisy Leakage Model

We note that FLR signature schemes in the bounded leakage model (as well as the CTL model) were given only very recently (in the standard model) by Malkin et al. [MTVY11] and Boyle et al.[BSW11]. However, these schemes are not secure in the *noisy leakage model*, formalized by Naor and Segev [NS09]. Noisy leakage is a realistic generalization of bounded leakage, in which the leakage is not necessarily of bounded length, and it is only guaranteed that the secret key still has some min-entropy even given the leakage. We note that our signature scheme, when instantiated with a hard relation secure in the noisy leakage model, is also secure in this model.

At a high level, constructions of reductions, from adversaries breaking unforgeability of known FLR signature schemes [MTVY11, BSW11] to underlying hard problems, rely on partitioning the message space into two parts - the *first* on which the reduction can generate signatures and the *second* on which it can not. These reductions break the underlying hard problem when all the adversary's signature queries come from the first partition while the forgery comes from the second partition. Further the signatures generated by the reduction on messages of first partition do not information theoretically fix the secret key. Therefore leakage of a signature from this partition would allow an adversary to break unforgeability without severely reducing the entropy of the secret key. Because of these reasons, the above scheme are not secure in the noisy leakage model.

On the other hand, in our scheme every signature information theoretically fixes the secret key. However, in the proof, a reduction can not answer the adversary's signature queries with these signatures that information theoretically fix the secret key. Our reduction solves this problem by providing "simulated signatures" instead which do not fix the secret key information theoretically, yet are computationally indistinguishable from the "real signatures." This allows us to achieve security in the noisy leakage model. We defer the details to the full version.

7 Leakage-Soundness and Simultaneous Leakage-Resilient ZK

7.1 Leakage-Sound Interactive Proofs

We now consider the opposite scenario where a malicious prover can obtain arbitrary leakage on the random coins of the verifier during the protocol execution. The question that we wish to investigate is whether it is possible to construct interactive proofs that remain sound even in such a scenario. Towards that goal, (as done previously) we model P and V as interactive turing machines that have the ability to flip coins during the protocol execution. At any point during the protocol execution, a malicious prover P^* may send a leakage query f (where $f(\cdot)$ is an arbitrary PPT length-decreasing function, described as a circuit) to the verifier. An honest verifier V, on receiving such a leakage query, computes f on her random coins used thus far in the protocol (i.e., the prover cannot leak on the future random coins of the verifier) and returns the output to the prover. In order to bound the leakage obtained by a cheating prover, we consider a leakage parameter ℓ and require that $|f(\cdot)| \leq \ell$ for every leakage query $f(\cdot)$.The prover may make any arbitrary polynomial number of leakage queries during the protocol execution, as long as the *total* leakage size is bounded by ℓ.

Informally speaking, we say that an interactive proof system is *leakage-sound* if it satisfies the soundness property even with respect to a cheating prover that can obtain leakage on the random coins of the verifier.

Definition 12 (Leakage-sound interactive proof system) *An interactive proof system $\langle P, V\rangle$ for a language $\mathcal{L}$ is said to be* ℓ-leakage-sound interactive proof system *if for every $x \notin \mathcal{L}$, and every interactive Turing machine P^* that makes any arbitrary polynomial number of leakage queries on the verifier's random coins (thus far in the protocol execution; in the manner as described above) such that*

the total leakage size is bounded by ℓ, the following holds:

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

If the soundness condition in the above definition is valid only against PPT Turing machines, then we say that $\langle P, V\rangle$ is a leakage-sound interactive *argument* system. We note that any public coin interactive proof system is already leakage-sound for any arbitrary amount of leakage from the verifier.

7.2 Simultaneous Leakage-Resilient Zero Knowledge

We finally consider the scenario where a cheating prover can obtain leakage on the random coins of an honest verifier while at the same time, a cheating verifier can obtain leakage on the honest prover's witness and random coins. We wish to investigate whether it is possible to construct an interactive proof system that *simultaneously* satisfies the two notions of leakage-soundness (c.f. Definition 12) and leakage-resilient zero knowledge (c.f. Definition 8). We call such an interactive proof system *simultaneous leakage-resilient zero knowledge*, as stated below formally.

Definition 13 (Simultaneous leakage-resilient zero knowledge proof system) *An interactive proof system $\langle P, V\rangle$ for a language $\mathcal{L}$ is said to be ℓ-*simultaneous leakage-resilient zero knowledge proof system *if it is ℓ-leakage-sound as per Definition 12 and leakage-resilient zero knowledge as per Definition 8.*

We note that our protocol presented in Figure 1 is already *simultaneous leakage-resilient zero knowledge*. In order to argue leakage soundness we start by observing that the commitment from the verifier to the prover is statistically hiding. At a high level this means that the commitment provided by the verifier can be opened to any value, and therefore leakage on the committed value and the randomness used in generating the commitment can be reduced to a leakage query (running possibly in unbounded time) on the message alone. In our protocol, the verifier provides a commitment to its challenge string. Therefore given the leakage, as long as at least $\omega(\log k)$ bit of entropy remains in the challenge string, soundness will be preserved. Finally, in order to achieve ℓ-leakage-soundness we will need to consider $\ell + \omega(\log k)$ repetitions of the Blum's protocol in our protocol presented in Figure 1.

Finally we note that, our construction of leakage resilient NIZKs (In Section 4) is *simultaneous leakage-resilient zero knowledge* for any arbitrary amount of leakage from the verifier. This follows trivially from the fact that in our construction of NIZKs the verifier is deterministic, and is not involved in any interaction.

8 Acknowledgements

Research supported in part from a DARPA/ONR PROCEED award, NSF grants 0916574 and 0830803, a Xerox Foundation Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant.

References

[ADN+10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In *EUROCRYPT*, pages 113–134, 2010.

[ADW09a] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[ADW09b] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In *ICITS*, pages 1–18, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[Ajt11] Miklos Ajtai. Secure computation with information leaking to an adversary. In *STOC*, 2011.

[AK96] Ross Anderson and Markus Kuhn. Tamper resistance: a cautionary note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–11, 1996.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[Bea96] Donald Beaver. Adaptive zero knowledge and computational equivocation (extended abstract). In *STOC*, pages 629–638, 1996.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition using super-polynomial simulation. In *Proc. 46th FOCS*, 2005.

[BSW11] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In *EUROCRYPT*, 2011.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge. In *Proc. 32th STOC*, pages 235–244, 2000.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In *EUROCRYPT*, pages 545–562, 2008.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437 (electronic), 2000. Preliminary version in STOC 1991.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO ' 2001*, pages 566–598, 2001.

[DGK+10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[DHLW10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DHLW10b] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proc.* 30*th STOC*, pages 409–418, 1998.

[DNW09] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties. In *TCC*, 2009.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[DPP97] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. *J. Cryptology*, 10(3):163–194, 1997.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *CRYPTO*, pages 526–545, 1989.

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge PCPs, and unconditional cryptography. In *CRYPTO*, pages 173–190, 2010.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–189, Summer 1996.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc.* 17*th STOC*, pages 291–304, 1985.

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, 2006.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *EUROCRYPT*, pages 339–358, 2006.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[HM96] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In *CRYPTO*, pages 201–215, 1996.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Extracting correlations. In *FOCS*, pages 261–270, 2009.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[Kat07] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[KP10] Eike Kiltz and Krzysztof Pietrzak. Leakage resilient elgamal encryption. In *ASIACRYPT*, pages 595–612, 2010.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[LLW11] Allison Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In *STOC*, 2011.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 179–188. ACM, 2009.

[LRW11] Allison Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, 2011.

[LZ09] Yehuda Lindell and Hila Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. In *TCC*, pages 183–201, 2009.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC*, pages 306–315, 2006.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[MS08] Tal Moran and Gil Segev. David and goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, pages 527–544, 2008.

[MTVY11] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In *EUROCRYPT*, 2011.

[MY04] Philip D. MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In *EUROCRYPT*, pages 382–400, 2004.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness (extended abstract). In *CRYPTO*, pages 128–136, 1989.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 21st STOC*, pages 33–43, 1989.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *CT-RSA*, pages 1–20, 2006.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *STOC*, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, 2002.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[Ros04] Alon Rosen. A note on constant-round zero-knowledge proofs for NP. In *TCC*, pages 191–202, 2004.

[Sah99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proc. 40th FOCS*, pages 543–553, 1999.

Appendix

A True Simulation-Extractable Leakage-Resilient NIZK

A tag-based simulation-extractable leakage-resilient (tSE-LR) NIZK argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ follows directly from the UC NIZK of Groth, Ostrovsky and Sahai [GOS06]. In fact it is (relatively) easier to construct tag-based tSE-LR-NIZK argument system. In our construction, we can use tags directly

while this was not feasible in the construction of UC-NIZK in [GOS06]. Also, unlike their construction, here we do not provide perfect security. A large part of the construction and the proof has been taken verbatim from [GOS06]. We provide the construction for $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ in this section. We start by listing the tools needed in the construction.

A.1 Tools

We will use a non-interactive zero knowledge argument system (K, P, V) with honest prover state reconstruction (cf. Definition 4 and Definition 5). Let (S_1, S_2, S_3) denote the simulator for (K, P, V). In addition, we use the following cryptographic tools.

Encryption with pseudorandom ciphertexts. A public-key cryptosystem $(K_{\text{pseudo}}, E, D)$ has pseudorandom ciphertexts of length $\ell_E(k)$ if for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} & \Pr\left[(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{E_{pk}(\cdot)}(pk) = 1\right] \\ \approx \quad & \Pr\left[(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{R_{pk}(\cdot)}(pk) = 1\right], \end{aligned} \tag{1}$$

where $R_{pk}(m)$ runs $c \leftarrow \{0,1\}^{\ell_E(k)}$ and every time returns a fresh c. We require that the cryptosystem has errorless decryption.

Trapdoor permutations over domain $\{0,1\}^{\ell_E(k)-1}$ imply pseudorandom cryptosystems as we can use the Goldreich-Levin hard-core bit [GL89] of a trapdoor permutation to make a one-time pad. Trapdoor permutations over $\{0,1\}^{\ell_E(k)-1}$ can for instance be constructed from the RSA assumption asusming $\ell_E(k)$ is large enough [CFGN96]. These can also be constructed from other special number theoretic assumptions as described in [GOS06].

Tag-based simulation-sound trapdoor commitment. A tag-based commitment scheme has four algorithms $(K_{\text{tag}-\text{com}}, \text{commit}, \text{Tcom}, \text{Topen})$. The key generation algorithm $K_{\text{tag}-\text{com}}$ produces a commitment key ck as well as a trapdoor key TK. There is a commitment algorithm that takes as input the commitment key ck, a message m and any tag $\texttt{tag}$ and outputs a commitment $c = \text{commit}_{ck}(m, \texttt{tag}; r)$. To open a commitment c with tag $\texttt{tag}$ we reveal m and the randomness r. Anybody can now verify $c = \text{commit}_{ck}(m, \texttt{tag}; r)$. As usual, the commitment scheme must be both hiding and binding.

In addition, to these two algorithms there are also a couple of trapdoor algorithms $\text{Tcom}, \text{Topen}$ that allow us to create an equivocal commitment and later open this commitment to any value we prefer. We create an equivocal commitment and an equivocation key as $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(\texttt{tag})$. Later we can open it to any message m as $r \leftarrow \text{Topen}_{\text{EK}}(c, m, \texttt{tag})$, such that $c = \text{commit}_{ck}(m, \texttt{tag}; r)$. We require that equivocal commitments and openings are indistinguishable from real openings. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} & \Pr\left[(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k) : \mathcal{A}^{\mathcal{R}(\cdot,\cdot)}(ck) = 1\right] \\ \approx \quad & \Pr\left[(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k) : \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(ck) = 1\right], \end{aligned} \tag{2}$$

where $\mathcal{R}(m, \texttt{tag})$ returns a randomly selected randomizer and $\mathcal{O}(m, \texttt{tag})$ computes $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(m, \texttt{tag})$; $r \leftarrow \text{Topen}_{\text{EK}}(c, m, \texttt{tag})$ and returns r. Both oracles ignore tags that have already been submitted once.

The tag-based simulation-soundness property means that a commitment using $\texttt{tag}$ remains binding even if we have made equivocations for commitments using different tags. For all non-uniform

polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\Big[(ck, \text{TK}) \leftarrow K(1^k); (c, \texttt{tag}, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(ck) : \texttt{tag} \notin Q \text{ and } c = \text{commit}_{ck}(m_0, \texttt{tag}; r_0) = \text{commit}_{ck}(m_1, \texttt{tag}; r_1) \text{ and } m_0 \neq m_1\Big] \approx 0, \tag{3}$$

where $\mathcal{O}(\text{commit}, \texttt{tag})$ computes $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(\texttt{tag})$, returns c and stores $(c, \texttt{tag}, \text{EK})$, and $\mathcal{O}(\text{open}, c, m, \texttt{tag})$ returns $r \leftarrow \text{Topen}_{ck}(\text{EK}, c, m, \texttt{tag})$ if $(c, \texttt{tag}, \text{EK})$ has been stored, and where Q is the list of tags for which equivocal commitments have been made by $\mathcal{O}$.

The term tag-based simulation-sound commitment comes from Garay, MacKenzie and Yang [GMY06], while the definition presented here is from MacKenzie and Yang [MY04]. The latter paper offers a construction based on one-way functions.

A.2 Construction of True Simulation-Extractable Leakage-Resilient NIZK

The issues that come up in the UC NIZK construction of Groth, Ostrovsky and Sahai [GOS06] also come up in our construction of true simulation-extractable leakage resilient NIZKs. The two key hurdles that come up in the construction are: First, the simulator $\mathcal{S}$ has to simulate the NIZK arguments (let Π be one of them) without knowing the witness. Furthermore, given the witness $\mathcal{S}$ must be able to simulate the randomness that would explain Π. $\mathcal{S}$ needs to do this in order to answer the leakage queries. The second problem is that if an adversary generates an acceptable NIZK argument Π for a statement C then $\mathcal{S}$ must use Π and output a witness w such that $C(w) = 1$.

The main idea in overcoming these hurdles is to commit to the witness w and make a NIZK argument with honest prove state reconstruction such that the commitment contains a witness w such that $C(w) = 1$. The honest prover state reconstruction property of the NIZK argument helps us to simulating the leakage queries. But, this leaves us with the commitment scheme. On one hand, when $\mathcal{S}$ simulates NIZK arguments we want to make equivocal commitments that can be opened arbitrarily since $\mathcal{S}$ does not know the witness and may need to answer leakage queries. On the other hand, when $\mathcal{S}$ sees an adversarially generated NIZK proof then we want to be able to extract the witness.

We construct such a commitment scheme, just like in [GOS06], from the tools specified in the previous section in a manner related to the construction of a UC commitment by Canetti et al. [CLOS02]. We use a tag-based simulation-sound trapdoor commitment scheme to commit to each bit of w. If w has length ℓ this gives us commitments $c_1, \ldots, c_\ell$. $\mathcal{S}$ can use the trapdoor key TK to create equivocal commitments that can be opened to arbitrary bits. This enables $\mathcal{S}$ to simulate the leakage queries made by the verifier.

We still have an extraction problem since $\mathcal{S}$ may not be able to extract a witness from tag-based commitments created by the adversary. To solve this problem we encrypt the openings of the commitments. Now $\mathcal{S}$ can extract witnesses, but we have reintroduced the problem of equivocation. In a simulated commitment there may be two different openings of a commitment c_i to respectively 0 and 1, however, if the opening is encrypted then we are stuck with one possible opening. This is where the pseudorandomness property of the cryptosystem comes in handy. $\mathcal{S}$ can simply make two ciphertexts, one containing an opening to 0 and one containing an opening to 1. Since the ciphertexts are pseudorandom, $\mathcal{S}$ can later open the ciphertext containing the desired opening and plausibly claim that the other ciphertext was chosen as a random string. To recap, the idea so far to commit to a bit b is to make a commitment c_i to this bit, and create a ciphertext $c_{i,b}$ containing an opening of c_i to b, while choosing $c_{i,1-b}$ as a random string.

The commitment scheme is once again equivocable, however, once again we must be careful that $\mathcal{S}$ can extract a message from an adversarial commitment during the simulation. We stress that this is not a problem as the adversary can not produce equivocable commitments using a tag different from the tags on which it gets commitments from $\mathcal{S}$.

The resulting protocol can be seen in Figure 5. We use the notation from Section A.1.

Common reference string generation:

1. $(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k)$
2. $(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k)$
3. $(\sigma, \tau) \leftarrow S_1(1^k)$
4. Return $\Sigma = (ck, pk, \sigma)$

Proof: On input (Σ, C, w) such that $C(w) = 1$ do

1. For $i = 1$ to ℓ select r_i at random and let $c_i := \text{commit}_{ck}(w_i, \texttt{tag}; r_i)$
2. For $i = 1$ to ℓ select R_{w_i} at random and set $c_{i,w_i} := E_{pk}(r_i; R_{w_i})$ and choose $c_{i,1-w_i}$ as a random string.
3. Let $c := (c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1})$
4. Create an NIZK proof π for the statement that there exists w and randomness such that c has been produced as described in steps 1,2 and 3 and $C(w) = 1$.
5. Return $\Pi = (\texttt{tag}, c, \pi)$

Verification: On input (Σ, C, Π)

1. Parse $\Pi = (\texttt{tag}, c, \pi)$
2. Verify the NIZK proof π
3. Return 1 if the check works out, else return 0.

Figure 5: Simulation Extractable Leakage Resilient NIZK argument $(\mathcal{K}, \mathcal{P}, \mathcal{V})$.

Theorem 4 *The protocol $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ described in Figure 5 is a true simulation-extractable leakage-resilient non-interactive zero knowledge argument system.*

Proof. Soundness and completeness of $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ follow directly from the soundness and completeness of the underlying NIZK. We are left to argue two things. First we need to argue that the protocol $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is leakage resilient non-interactive zero-knowledge. Secondly, we need to argue that we can extract a witness from a valid proof generated by an adversary.

SIMULATING Σ. $\mathcal{S}$ chooses the common reference string in the following way: It selects, $(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k)$; $(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k)$ and $(\sigma, \tau) \leftarrow S_1(1^k)$. It sets the CRS as $\Sigma := (ck, pk, \sigma)$. This means $\mathcal{S}$ is able to create and equivocate simulation-sound trapdoor commitments, decrypt pseudorandom ciphertexts and simulate NIZK proofs and later upon learning a witness simulate convincing randomness used for generating the proof.

SIMULATING PROOFS. $\mathcal{S}$ needs to simulate a proof that there exists w such that $C(w) = 1$, however, it may not use w. $\mathcal{S}$ uses $\texttt{tag}$ specified by $\mathcal{A}$ and forms ℓ equivocal commitments $(c_i, \text{EK}_i) \leftarrow \text{Tcom}_{\text{TK}}(\texttt{tag})$. $\mathcal{S}$ then simulates openings of the c_i's to both 0 and 1. For all $i = 1$ to ℓ and $b = 0$ to 1 it computes $\rho_{i,b} \leftarrow \text{Topen}_{\text{EK}_i}(c_i, b, \texttt{tag})$. It selects $r_{i,b}$ at random and sets $c_{i,b} := E_{pk}(\rho_{i,b}; r_{i,b})$. $\mathcal{S}$ sets $c := (c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1})$. Let x be the statement that there exists a witness w and randomness such that c has been correctly generated using w and $C(w) = 1$. $\mathcal{S}$ chooses randomness ρ and simulates the NIZK proof for x being true as $\pi \leftarrow S_2(\sigma, \tau, x; \rho)$. Let $\Pi = (\texttt{tag}, c, \pi)$ and return it as the simulated proof.

SIMULATING LEAKAGE. For any simulated proof Π generated by $\mathcal{S}$ it might need to answer a leakage query on the witness and randomness used to generate the proof Π. For this the simulator has access to a leakage oracle $L(\cdot)$.

We now describe how given the witness $\mathcal{S}$ can simulate the randomness that would lead P_i to produce such an proof Π. Since $\mathcal{S}$ created $c_i, c_{i,0}, c_{i,1}$ such that $c_{i,0}$ contains a 0-opening of c_i and $c_{i,1}$ contains a 1-opening of c_i it can produce good looking randomness to claim that the party committed to w_i. This gives us convincing randomness for constructing all these commitments and for producing the ciphertext c. $\mathcal{S}$ can now run the simulator algorithm S_3 to simulate randomness that would lead the prover to have produced the proof π. Hence any leakage query made on the witness and the randomness can be reduced to a leakage query made just on the witness and the simulator can use the leakage oracle to answer that query.

EXTRACTION. For an adversarially generated valid proof Π, $\mathcal{S}$ must extract a witness w. $\mathcal{S}$ parses c as $c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1}$. Since $\mathcal{S}$ knows the decryption key DK, it can then decrypt all $c_{i,b}$. This gives $\mathcal{S}$ plaintexts $\rho_{i,b}$. It checks for each i whether $c_i = \text{commit}_{ck}(b, \texttt{tag}; \rho_{i,b})$ and in that case b is a possible candidate for the i-th bit of w.

If successful in all of this, $\mathcal{S}$ lets w be these bits. However, if any of the bits are ambiguous, *i.e.*, w_i could be both 0 and 1, or if any of them are inextractable, then $\mathcal{S}$ outputs `fail`.

We will later argue that the probability of the NIZK argument Π being valid, yet $\mathcal{S}$ not being able to extract a witness is negligible.

HYBRIDS. We wish to argue that no PPT adversarial verifier $\mathcal{A}$ can distinguish between its interaction with a real prover and its interaction with the simulator $\mathcal{S}$. In order to do so we define several hybrid experiments and show that $\mathcal{A}$ cannot distinguish between any of them. Then we argue that our simulator [12] $\mathcal{S}$ can in fact also extract the witness from a valid proof generated by $\mathcal{A}$. We will now give the full description of the hybrid experiments and the security proof.

H1: This is real interaction between adversary $\mathcal{A}$ and $\mathcal{S}$. $\mathcal{S}$ obtains the witness for every theorem it proves. It use the witness in an honest way and answers leakage queries honestly as well.

H2: We modify H1 in the way $\mathcal{S}$ creates tag-based simulation-sound trapdoor commitments $c_1, \ldots, c_\ell$ to the bits of the witness. Let `tag` be the tag specified by the adversary. Instead of creating c_i by selecting r_i at random and setting $c_i = \text{commit}_{ck}(w_i, \texttt{tag}; r_i)$, we create an equivocal commitment $(c_i, \text{EK}_i) \leftarrow \text{Tcom}_{\text{TK}}(\texttt{tag})$ and subsequently produce randomness $\rho_{i,w_i} \leftarrow \text{Topen}_{\text{EK}_i}(c_i, w_i, \texttt{tag})$. We continue the proof using ρ_{i,w_i} instead of r_i.

H1 and H2 are indistinguishable because it is hard to distinguish tag-based commitments and their openings from tag-based equivocal commitments and their equivocations to the same messages (Equation (2)).

H3: In H3, we make another modification to the procedure followed by $\mathcal{S}$. We are already creating c_i as an equivocal commitment and equivocating it with randomness ρ_{i,w_i} that would open it to contain w_i. We run the equivocation procedure once more to also create convincing randomness that would explain c_i as a commitment to $1 - w_i$. This means, we compute $\rho_{i,1-w_i} \leftarrow \text{Topen}_{\text{EK}_i}(c_i, 1 - w_i, \texttt{tag})$. Instead of selecting $c_{i,1-w_i}$ as a random string, we choose to encrypt $\rho_{i,1-w_i}$ as $c_{i,1-w_i} = E_{pk}(\rho_{i,1-w_i}; r_{i,1-w_i})$ for a randomly chosen $r_{i,1-w_i}$. We still pretend that $c_{i,1-w_i}$ is a randomly chosen string when we carry out the NIZK proof π or when the leakage queries need to be answered.

H2 and H3 are indistinguishable because of the pseudorandomness property of the cryptosystem, see Equation (1). Suppose we could distinguish H2 and H3, then we could distinguish between an encryption oracle and an oracle that supplies randomly chosen strings.

[12] Note that our $\mathcal{S}$ is playing the role of the extractor $\mathcal{E}$ as well.

H4: Instead of making NIZK arguments using honest prover strategy we use the zero-knowledge with honest prover state reconstruction simulators. We use $\pi \leftarrow S_2(\sigma, \tau, \cdot; \rho)$ with ρ random to simulate the honest provers' NIZK arguments that c has been correctly generated. Finally, on input the witness we can use $r \leftarrow S_3(\sigma, \tau, x, \pi, \cdot, \rho)$ to create convincing randomness that would make the prover output π on the witness for c being correctly generated. So any leakage query on the randomness and the witness can be reduced to a leakage query of the witness alone.

The zero-knowledge with honest prover state reconstruction property of the NIZK proof implies that H3 and H4 are indistinguishable.

Simulation. Note that the simulator $\mathcal{S}$ in hybrid H4 already simulates the view of the adversary $\mathcal{A}$ in a way that is indistinguishable from its view while interacting with honest prover. This concludes the proof that the simulator $\mathcal{S}$ correctly simulates the view of the adversary. Now we need to argue that if the adversary $\mathcal{A}$ can in fact output a valid proof Π with a tag $\mathtt{tag}$ such that $\mathcal{S}$ never gave a proof using the tag $\mathtt{tag}$ then we can use the proof to extract a witness. Consider the following subsequent hybrids.

H5: Again, we look at the adversarially generated NIZK argument $\Pi = (\mathtt{tag}, c, \pi)$ for some C. Parse c as $c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1}$. Then we use the decryption key DK to attempt to decrypt the $c_{i,b}$'s to get $\rho_{i,b}$ so $c_{i,b} = \mathrm{commit}_{ck}(b, \mathtt{tag}; \rho_{i,b})$. We output $\mathtt{failure}$ if we encounter a $c_i = \mathrm{commit}_{ck}(0, \mathtt{tag}, \rho_{i,0}) = \mathrm{commit}_{ck}(1, \mathtt{tag}, \rho_{i,1})$.

Tag-based simulation-soundness, see Equation (3), of the commitment scheme implies that H4 and H5 are indistinguishable. To see this consider the tag $\mathtt{tag}$. Outputting $\mathtt{failure}$ corresponds to breaking the binding property of the commitment scheme, unless we have previously created an equivocal commitment with tag $\mathtt{tag}$. But we already ruled out that possibility.

H6: As in H5, we try to extract $\rho_{i,0}, \rho_{i,1}$'s. We output $\mathtt{failure}$ if there is an i such that we cannot decrypt either $c_{i,0}$ or $c_{i,1}$ to give us $\rho_{i,b}$ so $c_i = \mathrm{commit}_{ck}(b, \mathtt{tag}; \rho_{i,b})$. We ruled out the possibility of both $\rho_{i,0}$ and $\rho_{i,1}$ being an opening of c_i in H5, so if everything is OK so far we have a uniquely defined w such that for all i we have $c_i = \mathrm{commit}_{ck}(w_i, \mathtt{tag}; \rho_{i,w_i})$. We output $\mathtt{failure}$ if $C(w) \neq 1$.

Call c well-formed if $c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1}$ are such that for all $i = 1$ to ℓ at least one of the $c_{i,0}, c_{i,1}$ will have a proper $\rho_{i,b}$ so $c_i = \mathrm{commit}_{ck}(b, \mathtt{tag}; \rho_{i,b})$, and if all of these openings are unique then the bits constitute a witness w for $C(w) = 1$. Observe, from the soundness of NIZK [13] it follows that with overwhelming probability c is well-formed and we have negligible chance of outputting $\mathtt{failure}$. This means H5 and H6 are indistinguishable.

Extraction. Observe that $\mathcal{S}$ in H6 has already obtained a witness w corresponding to the the valid proof Π generated by the adversary $\mathcal{A}$. Our simulator can output this as its output and this concludes the proof that the NIZK argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is indeed simulation extractable. □

Remark on common random string. We note that in our scheme the CRS consists of three components. It consists of a public key of a pseudorandom encryption scheme, a public key of a tag-based simulation sound trapdoor commitment scheme and a CRS for the underlying NIZK proof (as explained in Section A.1). We stress that actually all these components can be chosen randomly, i.e., the sampled without actually learning the associated secret parameters. As explained in [GOS06], we can construct public-key encryption with pseudorandom ciphertexts under the decisional linear

[13] Groth et. al. [GOS06] argue that it is problematic if the language about which the theorem is being proved is chosen depending on the CRS. We ignore this as this does not affect our application. However it can be noted that the same argument holds for our NIZK as well.

assumption. A public key of such a scheme consists of three random generators of a prime order group which can be sampled without the knowledge of the corresponding secret values. As noted in [MY04] tag-based simulation-sound commitment scheme can be constructed using a signature scheme and we know a number of signature schemes in which the public key can be sampled without the knowledge of the secret key. Brent's signature scheme serves as one such example in the setting of bilinear groups. Finally, as noted in [GOS06], the CRS for the underlying NIZK proof system can be chosen to be a common random string at the cost of having a proof system that is only statistically sound, which suffices in our setting. In summary, we have argued that the CRS in our scheme can be common random string.

B Impossibility of LR-ZK for $\lambda < 1$

Theorem 5 *There exists a language $\mathcal{L}$ such that there exists no interactive proof system $\langle P, V\rangle$ that is λ-*leakage-resilient zero knowledge *where $\lambda < 1$.*

Proof Sketch. Consider a very simple language $\mathcal{L}$ that consists of every string $x \in \{0,1\}^*$. The witness relation $\mathcal{R}$ associated with $\mathcal{L}$ consists of pairs (x, w) such that for a given instance x, every string $w \in \{0,1\}^{|x|}$ is a witness. In this setting we will construct an adversarial verifier V^* and a distinguisher D such that D, that gets the prover's witness as auxiliary input, can distinguish between $\text{view}_{V^*}(x, z)$ and $\mathcal{S}^{L_w^{k,\lambda}(\cdot)}(x, z)$ with non-negligible probability.

Consider the scenario where for a given instance x, the prover's witness w is sampled uniformly at random among all possible witnesses. Consider a V^* that works as follows. It makes a leakage query that leaks the whole witness.[14] Finally V^* outputs the leaked witness as part of its view. The construction of D is straight forward. D gets the prover's witness w as auxiliary input. It outputs 1 if the view of V^* contains w and 0 otherwise.

It is easy to see that there's no way that the simulator can output the correct witness every time when at most $\lambda \cdot |w|$ bits of leakage are available to it. An easy way to argue this is as follows: fix the random coins of the simulator, but keep the randomization over the witnesses. Then note that the simulator will get the witness wrong at least 1/2 the time, for any fixed random tape. Therefore, averaging over his random tapes, the simulator must still get it wrong at least 1/2 the time. Hence, the distinguisher D will be able to distinguish between $\text{view}_{V^*}(x, z)$ and $\mathcal{S}^{L_w^{k,\lambda}(\cdot)}(x, z)$ with non-negligible probability. □

Remark 1 *In the proof sketch we gave the proof for a trivial language where every string was in the language and every string was a witness. We stress that this was only done for simplicity. We can construct an NP-complete language as well and argue in a similar way. Finally, the impossibility holds even in case of bounded leakage as well by an analogous argument.*

C Leakage-Resilient Zero Knowledge using Pre-Processing

In this subsection we argue that it would be difficult to construct a leakage resilient zero knowledge proof system in a setting where there is a "leakage-free" pre-processing phase prior to the actual protocol execution, but the simulator does not have any access to a leakage oracle (unlike our model). In order to establish our argument, we will assume that it is not possible for a simulator to *reverse-engineer* the leakage queries of an adversarial verifier (or in other words, it is possible for an adversarial

[14] Note that leakage of the entire witness is not necessary for the proof to work. In particular, it is easy to see that the proof works even with partial leakage of the witness.

verifier to *obfuscate* its leakage queries). Consider a language $\mathcal{L}$ and a prover $P = (P_1, P_2)$ in the protocol $\langle P, V \rangle$ that wants to prove that $x \in \mathcal{L}$. Let w denote the witness that is given to P as private input. Before the start of the actual protocol, P_1 runs a private "leakage-free" pre-processing phase on w to generate a valid witness w' for an instance $x' \in \mathcal{L}'$. The new witness w' is given as input to P_2. P_2 now interacts with the verifier and attempts to prove that $x' \in \mathcal{L}'$.

Note that in order to argue the correctness and soundness of $\langle P, V \rangle$, we will need that $x' \in \mathcal{L}'$ if and only if $x \in \mathcal{L}$. Now, since the simulator will not have access to a valid witness w, it will not have access to w' as well. However, a cheating verifier may simply make a leakage query that checks if w' is indeed a valid witness for x' and encrypts the output (under a secret key known only to the verifier). Now assuming that the simulator can not reverse engineer the leakage query, simulator will not be able to respond to the query correctly (since otherwise, we can contradict the soundness of the zero knowledge proof system).

The argument presented above makes strong unproven assumptions and might not seem satisfactory for that reason. Nonetheless, we stress that our goal here is not to obtain a strong impossibility result in this direction but rather to highlight the fact that this direction is not promising and hence not worth pursuing in the interactive setting when strong guarantees against leakage attacks are desired.