Achieving Constant Round Leakage-Resilient Zero-Knowledge

Omkant Pandey*
`omkant@cs.ucla.edu`

Abstract

Recently there has been a huge emphasis on constructing cryptographic protocols that maintain their security guarantees even in the presence of side channel attacks. Such attacks exploit the physical characteristics of a cryptographic device to learn useful information about the internal state of the device. Designing protocols that deliver meaningful security even in the presence of such leakage attacks is a challenging task.

The recent work of Garg, Jain, and Sahai formulates a meaningful notion of zero-knowledge in presence of leakage; and provides a construction which satisfies a weaker variant of this notion called $(1+\epsilon)$-leakage-resilient-zero-knowledge, for every constant $\epsilon > 0$. In this weaker variant, roughly speaking, if the verifier learns ℓ bits of leakage during the interaction, then the simulator is allowed to access $(1+\epsilon)\cdot\ell$ bits of leakage. The round complexity of their protocol is $\lceil \frac{n}{\epsilon} \rceil$.

In this work, we present the first construction of leakage-resilient zero-knowledge satisfying the ideal requirement of $\epsilon = 0$. While our focus is on a feasibility result for $\epsilon = 0$, our construction also enjoys a constant number of rounds. At the heart of our construction is a new "public-coin preamble" which allows the simulator to recover arbitrary information from a (cheating) verifier in a "straight line." We use non-black-box simulation techniques to accomplish this goal.

1 Introduction

The concept of zero-knowledge interactive proofs, originating in the seminal work of Goldwasser, Micali, and Rackoff [GMR85], is a fundamental concept in theoretical cryptography. Informally speaking, a zero-knowledge proof allows a prover P to prove an assertion x to a verifier V such that V learns "nothing more" beyond the validity of x. The proof is an interactive and randomized process. To formulate "nothing more," the definition of zero-knowledge requires that for every malicious V^* attempting to lean more from the proof, there exists a polynomial time simulator S which on input *only* x, simulates a "real looking" interaction for V^*.

In formulating the zero-knowledge requirement, it is assumed that the prover P is able to keep its internal state — the witness and the random coins — perfectly hidden from the verifier V^*. It has been observed, however, that this assumption may not hold in many settings where an adversary has the ability to perform *side channel attacks.* These attacks enable the adversary to learn useful information about the internal state of a cryptographic device (see e.g., [Koc96, AK97, QS01, OST06] and the references therein). In presence of such attacks, standard cryptographic primitives often fail to deliver any meaningful notion of security.

To deliver meaningful security in the presence side channel attacks, many recent works consider stronger adversarial models in which the device implementing the honest algorithm *leaks* information about its internal state to the adversary. The goal of these works is then to construct cryptographic

*Visiting Researcher, Microsoft Research, Bangalore (India).

primitives that are "resilient" to such leakage. Leakage resilient constructions for many basic cryptographic tasks are now known [DP08, AGV09, Pie09, DKL09, ADW09a, ADW09b, NS09, KP10, BKKV10, DHLAW10b, DHLAW10a, LW10, FRR$^+$10, LRW11, BSW11, Ajt11].

Leakage-resilient zero-knowledge. Very recently Garg, Jain, and Sahai [GJS11] initiated an investigation of leakage-resilient zero-knowledge (LRZK). Their notion considers a cheating verifier V^* who can learn an arbitrary amount of leakage on the internal state of the honest prover. This is formulated by allowing V^* to make leakage queries $F_1, F_2, \ldots$ throughout the execution of the protocol. Then the definition of LRZK, roughly speaking, captures the intuition that no such V^* learns anything *beyond the validity of the assertion and the leakage.*

The actual formulation of this intuition is slightly more involved. Observe that during the simulation, S will need to answer leakage queries of V^*, which may release information about the witness to V^*. Simulator S cannot answer such queries without having access to the witness. The definition of [GJS11] therefore provides S with access to *leakage oracle* which holds a witness to x. The oracle, $\mathcal{L}_w^n(\cdot)$, is parameterized by the witness w and $n = |x|$; on input a function F expressed as a boolean circuit, it returns $F(w)$. To ensure that S can answer leakage requests of V^*, the simulator is allowed to query $\mathcal{L}_w^n$ on leakage functions of its choice. Of course, providing S with uncontrolled access to the witness will render the notion meaningless.[1] Therefore, to ensure that the notion delivers meaningful security, the LRZK definition requires the following restriction on the length of bits that S can read from $\mathcal{L}_w^n$. Suppose that $S^{\mathcal{L}_w^n}$ outputs a simulate view υ for V^*. Denote by $\ell_S(\upsilon)$ the number of bits S reads from $\mathcal{L}_w^n$ in generating this particular view υ. Denote by $\ell_{V^*}(\upsilon)$ the total length of the leakage answers that S provides to V^* (which are already included in υ). Then, it is required that:

$$\ell_S(\upsilon) \leq \ell_{V^*}(\upsilon). \tag{1}$$

More precisely, in [GJS11], a slightly more general notion of $(1+\epsilon)$-LRZK is defined in which the above condition is relaxed to:

$$\ell_S(\upsilon) \leq (1+\epsilon) \cdot \ell_{V^*}(\upsilon),$$

where $\epsilon > 0$ is a constant. [GJS11] show that despite this relaxation, $(1+\epsilon)$-LRZK still delivers meaningful security by applying it to weaken assumptions on hardware tokens based cryptography. They also provide a construction of a protocol (for all of $\mathcal{NP}$) satisfying $(1+\epsilon)$-LRZK for every a-priori fixed constant $\epsilon > 0$. The round complexity of their protocol is $\lceil \frac{n}{\epsilon} \rceil$.

Although $(1+\epsilon)$-LRZK delivers a meaningful notion, it is much weaker than LRZK (i.e., $\epsilon = 0$, equation (1)). Allowing a positive ϵ enables the simulator S to learn strictly more than what a cheating verifier would learn from leakage. This can be particularly problematic in protocol composition [Can00, Can01] since using such a simulator in place of a cheating party will result in making more than "allowed" output queries to the ideal functionality.

The main result. In this work, we present the first construction of an LRZK protocol satisfying $\epsilon = 0$. Although our main goal is to obtain a feasibility result, our protocol also enjoys a *constant* number of rounds. Our protocol uses standard cryptographic tools. However, it requires some of them – particularly, oblivious transfer – to have an *invertible sampling* property [DN00, IKOS10]. To the best of our knowledge, instantiations satisfying this property are known only based on the decisional Diffie-Hellman assumption (DDH) [DH76]. We leave constructing an LRZK proof system based on general assumption as an interesting open question.

[1] S can simply access the entire witness, and then simulate.

Theorem 1 (Main Result)**.** *Suppose that the decision Diffie-Hellman assumption holds. Then, there exists a constant-round leakage-resilient zero-knowledge proof system for all languages in $\mathcal{NP}$.*

We remark that the low round-complexity is usually a desirable protocol feature [GK96, Bar01, Ros00, PRS02, Ros04]. In the context of side channel attacks, however, it can be particularly attractive. This is because a protocol with high round complexity requires the device to maintain state for more rounds, and perhaps for longer periods. This, in turn, can provide a side-channel adversary with more opportunities to attack the device.

1.1 An Overview of Our Approach

Let us start by recalling the main difficulty in constructing an LRZK protocol. Recall that a zero-knowledge simulator S "cheats" in the execution of the protocol so that it can produce a convincing view. When dealing with leakage, not only the simulator needs to continue executing the protocol, but it also needs to "explain its actions" so far by maintaining a state consistent with an honest prover.

To be able to simultaneously perform these two actions, the GJS simulator does the following. It combines the following two different but well-known methods of "cheating." The first method, due to Goldreich-Kahan [GK96], requires the verifier to commit its challenge ch; the second method, due to Feige-Shamir [FS89], requires the prover to use equivocal[2] commitments. The GJS simulator uses these methods *together.* It uses ch to perform its main simulation (by using [GK96] strategy), and uses the trapdoor of equivocal commitments, denoted t_1, to "explain its actions" so far. We call (t_1, ch) the double trapdoor.

The GJS simulator "rewinds" the verifier to obtain the two trapdoors before it actually enters the main proof stage. By using a precise rewinding strategy [MP06], GJS achieves $(1+\epsilon)$-LRZK. However, since rewinding strategy is crucial to their simulation, this approach by itself seems insufficient for achieving LRZK.

A fundamentally different simulation strategy, in which the simulator uses the program of the malicious verifier V^*, was presented in Barak's work [Bar01]. This method does not need to "rewind" the verifier to produce its output. Our first idea there is to somehow use Barak's simulation strategy along with the use of equivocal commitments as in [FS89]. Unfortunately, this does not work since the trapdoor t_1 for equivocation has to be somehow recovered and only then any other simulation strategy (such as knowing the challenge ch) can be used.

We therefore modify this approach so that we can use Barak's method to recover arbitrary information from the verifier during the simulation. For the purpose of this discussion, let us assume that Barak's technique provides a way for P and V to interactively generate a statement σ for some $\mathcal{NP}$-relation $\mathbf{R}_{sim}$ so that no cheating prover P^* can prove $\sigma \in \mathbf{L}_{sim}$, but a simulator S holding the program of the cheating verifier V^* will always have a witness ω such that $\mathbf{R}_{sim}(\sigma, \omega) = 1$. At this point, let us just assume that the verifier does not ask leakage queries.

Then, we need to design a two party protocol for the following task. The first party P holds a private input ω, the second party V holds an arbitrary private message m, the common input to both parties is σ. The protocol allows P to learn m if and only if $\mathbf{R}_{sim}(\sigma, \omega) = 1$, nothing otherwise; V learns nothing. This is similar in spirit to the *conditional disclosure* primitive of [GIKM00], except that here the condition is an arbitrary $\mathcal{NP}$-relation $\mathbf{R}_{sim}(\sigma, \omega) = 1$. Constructing such protocols for $\mathcal{NP}$-conditions has not been studied, since they follow from work on secure two-party computation [Yao82, GMW87, MR91]. Clearly, we cannot directly use secure two-party computation since their

[2]These are commitments which, given appropriate trapdoor information, can be opened to both 0 or 1.

security-guarantee is often *simulation-based* — which we do not know how to do under leakage attacks.

Our second key observation is that we do not really require the strong simulation-based guarantee. We only need to construct a conditional disclosure protocol for a very specific $\mathcal{NP}$-relation. We construct such a protocol based on Yao's garbled circuit technique. We show that if we use properly chosen OT protocols (constructed in [AIR01, NP01]) — then we get a conditional disclosure protocol. In addition, the protocol ensures that the messages of P are pseudorandom (more precisely, invertible samplable [DN00, IKOS10]). As a result, the protocol maintains its security claims even in the presence of leakage. This is a two-round protocol, and a crucial ingredient in achieving leakage resilience.

Armed with this new tool, simulation now seems straightforward: use the conditional disclosure protocol to recover both (t_1, ch) and then use the GJS-simulator. While this idea works, there is an issue with proving the *soundness* of this protocol. Recall that in Barak's protocol, one must find collisions in the hash function h to prove that no cheating P^* can succeed in learning a witness to statement σ. With our current ideas, this strategy for proving soundness fails.

Our third key observation is to modify Barak's statement generation protocol so that we can perform extraction during the process of statement generation. Assuming some familiarity with Barak's protocol, this can be achieved by using an extractable commitment scheme in the universal-argument (UARG) phase. However, using an extractable commitment scheme will prevent us from answering leakage queries and mess up the entire simulation. We therefore use a very special type of commitment scheme constructed by Barak-Lindell [BL04]. We observe that a minor modification of their protocol ensures that the messages sent by the committer are *pseudorandom* strings that are invertible samplable [DN00, IKOS10]. By using this commitment protocol in the preamble phase, we can prove the soundness of our protocol.

Recall that we work in the model of [GJS11]. In this model the verifier is allowed to ask arbitrary leakage queries $F_1, F_2, \ldots$ on prover's state. The state of the prover at any given round only consists of its witness and the randomness up to that round. In particular, the randomness of future rounds is determined only at the beginning of those round. Observe that all ingredients described by us so far actually require the prover to send only random strings. Therefore, it is easy to asnswer the leakage queries up to this point in the simulation. By the time simulator enters the main body, it recovers (t_1, ch) and use them to answer leakage queries as in [GJS11].

1.2 Related Work

Relevant to our work the zero-knowledge proofs in other more complex attack models such as man-in-middle attacks [DDN91], concurrent attacks [DNS98], resettable attacks [CGGM00, BGGL01], and so on. Also relevant to our work are different variants of non-black-box simulation used in the literature [Bar01, Bar02, Pas04, PR05, DGS09] as well as efficient and universal arguments [Kil92, Mic94, BG02, IKO07].

The explicit study of leakage-resilient cryptography was started by Dziembowski and Pietrzak [DP08]. Related study on protecting devices appears in the works of Ishai, Prabhakaran, Sahai, and Wagner [ISW03, IPSW06]. After these works a long line line of research has focussed on constructing primitives resilient to leakage including public-key encryption and signatures [AGV09, Pie09, DKL09, ADW09a, ADW09b, NS09, KP10, BKKV10, DHLAW10b, DHLAW10a, LRW11, BSW11, LW10], devices [FRR$^+$10, Ajt11], and very recently interactive protocols [GJS11, BCH12].

Also relevant to our work are works on adaptive security [CFGN96] and invertible sampling [DN00, IKOS10]. Adaptively secure protocols and leakage-resilience in interactive protocols is tightly connected [BCH12].

2 Notation and Definitions

Notation. For a randomized algorithm A we write $A(x;r)$ the process of evaluating A on input x with random coins r. We write $A(x)$ the process of sampling a uniform r and then evaluating $A(x;r)$. We define $A(x,y;r)$ and $A(x,y)$ analogously. The set of natural numbers is represented by $\mathbb{N}$. Unless specified otherwise, $n \in \mathbb{N}$ represents a security parameter available as an implicit input when necessary. All inputs are assumed to be of length at most polynomial in n. We assume familiarity with standard concepts such as interactive Turing machines (ITM), computational indistinguishability, commitment schemes, $\mathcal{NP}$-languages, witness relations and so on (see [Gol04]).

For two randomized ITMs A and B, we denote by $[A(x,y) \leftrightarrow B(x,z)]$ the interactive computation between A and B, with A's inputs (x,y) and B's inputs (x,z), and uniform randomness; and $[A(x,y;r_A) \leftrightarrow B(x,z;r_B)]$ when we wish to specify randomness. We denote by $\text{VIEW}_B[A(x,y) \leftrightarrow B(x,z)]$ and $\text{OUT}_B[A(x,y) \leftrightarrow B(x,z)]$ the view and output of B in this computation; $\text{VIEW}_A, \text{OUT}_A$ are defined analogously. Finally, $\text{TRANS}[A(x,y) \leftrightarrow B(x,z)]$ denotes the public transcript of the interaction $[A(x,y) \leftrightarrow B(x,z)]$.

For two probability distributions D_1 and D_2, we write $D_1 \overset{c}{\equiv} D_2$ to mean that D_1 and D_2 are computationally indistinguishable.

Definition 1 (Interactive Proofs)**.** *A pair of probabilistic polynomial time interactive Turing machines $\langle P, V\rangle$ is called an interactive proof system for a language $L \in \mathcal{NP}$ with witness relation R if the following two conditions with respect to some negligible function* negl$(\cdot)$*:*

- *Completeness: for every $x \in L$, and every witness w such that $R(x,w) = 1$,*

$$\Pr\left[\text{OUT}_V[P(x,w) \leftrightarrow V(x)] = 1\right] \geq 1 - \text{negl}(|x|).$$

- *Soundness: for every $x \notin L$, every interactive Turing machine P^*, and every $y \in \{0,1\}^*$,*

$$\Pr\left[\text{OUT}_V[P^*(x,y) \leftrightarrow V(x)] = 1\right] \leq \text{negl}(|x|).$$

If the soundness condition holds only against polynomial time machines P^*, $\langle P, V\rangle$ is called an argument system. We will only need/construct argument systems in this work.

Leakage attack. Machine P and V are modeled as randomized ITM which interact in rounds. It is assumed that the the random coins used by a party in any particular round are determined only at the beginning of that round. Denote by `state` a variable initialized to prover's private input w. At the end beginning of each round i, P flips coins r_i to be used for that round, and updates `state` $:=$ `state` $\|\ r_i$. A leakage query on prover's state in round i corresponds to verifier sending a function F_i (represented as a polynomial-sized circuit), to which the prover responds with $F_i(\texttt{state})$. The verifier is allowed to any number of arbitrary leakage queries throughout the interaction. A malicious verifier who obtains leakage under this setting is said to be launching a *leakage attack.*

To formulate zero-knowledge under a leakage attack, we consider a PPT machine S called the simulator, which receives access to an oracle $\mathcal{L}_w^n(\cdot)$. $\mathcal{L}_w^n(\cdot)$ is called the leakage oracle, and parametrized by the witness w and the security parameter n. A query to the leakage oracle consists of an efficiently computable function F, to which the oracle responds with $F(w)$. The leakage-resilient zero-knowledge is defined by requiring that the output of S be computationally indistinguishable from the real view; in addition the length of all bits read by S from $\mathcal{L}_w^n$ in producing a particular view v is at most the length of leakage answers contained in the v.

For $x \in L$, w such that $R(x,w)=1$, $z \in \{0,1\}^*$, and randomness $r \in \{0,1\}^*$ defining the output $v = S^{\mathcal{L}^n_w(\cdot)}(x,z;r)$, we let the function $\ell_{S,r}(v)$ denote the number of bits that S receives from $\mathcal{L}^n_w(\cdot)$ in generating view v with randomness r. Further, we let the function $\ell_{V^*}(v)$ denote the total length of leakage answers that V^* receives in the output v. By convention, randomness r will be included only when we need to be explicit about it.

Definition 2 (Leakage-resilient Zero-knowledge)**.** *We say that an interactive proof system $\langle P, V\rangle$ for a language $L \in \mathcal{NP}$ with a witness relation R, is leakage-resilient zero-knowledge if for every probabilistic polynomial time machine V^* launching a* leakage attack *on P, there exists a probabilistic polynomial time machine S such that the following two conditions hold:*

1. *For every $x \in L$, every w such that $R(x,w)=1$, and every $z \in \{0,1\}^*$, distributions $\text{VIEW}_{V^*}[P(x,w) \leftrightarrow V^*(x,z)]$ and $S^{\mathcal{L}^n_w(\cdot)}(x,z)$ are computationally indistinguishable.*
2. *For every $x \in L$, every w such that $R(x,w)=1$, every $z \in \{0,1\}^*$, and every sufficiently long $r \in \{0,1\}^*$ defining the output $v = S^{\mathcal{L}^n_w(\cdot)}(x,z;r)$, it holds that $\ell_S(v,r) \leq \ell_{V^*}(v)$.*

The definition of standard zero-knowledge is obtained by removing condition 2, and enforcing that no leakage queries are allowed to any machine.

3 Cryptographic Tools

We start by recalling some standard cryptographic tools and two-party protocols. Looking ahead, we will require that our protocols satisfy the following important property. For a specific party (chosen depending upon the protocol), all messages sent by this party be pseudorandom strings. In some cases where this is not possible, it will be sufficient if the messages are pseudorandom elements of group (e.g., a prime-order subgroup of $\mathbb{Z}_p^*$ for a (safe) prime p of length n).[3] We will provide necessary details when appropriate.

Statistically-binding commitments. We use Naor's scheme [Nao89], based on a pseudorandom generator (prg). Recall that in this scheme, first the receiver sends a random string τ of length $3n$; to commit to bit b, the sender selects a uniform seed s of length n and sends y such that if $b=0$ then $y = \mathsf{prg}(s)$, otherwise $y = \tau \oplus \mathsf{prg}(s)$. This scheme is statistically binding; in addition, *sender's message is pseudorandom.* A string can be committed by committing bitwise, and it suffices to use same τ for all the bits. We write $\mathsf{sbcom}_\tau(m;s)$ to represent sender's string, when receiver's first message is τ.

Statistically-hiding commitments. We use a statistically hiding commitment scheme as well. We require the receiver of this scheme to be *public coin.* Such schemes are known, including a two-round string commitment scheme, based on collision-resistant hash functions (crhf) [NY89, HM96, DPP97]. We write $\mathsf{shcom}_\rho(m;s)$ to denote sender's commitment string, when receiver's first message is ρ. Without loss of generality, $|\rho| = n$.

[3]This will be sufficient since public sampling from such a group admits *invertible sampling* [DN00, IKOS10]. However, it is more convenient to directly work with the assumption that algorithms can receive random elements in such a group as part of their random tape.

Zero-knowledge proofs. We use a statistical zero-knowledge argument-of-knowledge (szkaok) protocol for proving $\mathcal{NP}$-statements. We require the verifier of this protocol to be *public coin.* Such protocols are known to exist; including a constant-round protocol based on crhf [Bar01, BG02, PR05], and a $\omega(1)$-round protocol based on statistically-hiding commitments [GMW91, Blu87].

We choose the constant-round protocol of Pass and Rosen, denoted Π_{PR}, as our candidate szkaok. Let S_{PR} denote the corresponding simulator for Π_{PR}. We remark that S_{PR} is a "straight-line" simulator, with strict polynomial running time.

3.1 Oblivious Transfer

We will use a *two-round* oblivious transfer protocol $\mathsf{OT} := \langle S_{\text{OT}}, R_{\text{OT}} \rangle$. For the choice bit b of the receiver, we denote by $\{R_{\text{OT}}(1^n, b)\}_{n \in \mathbb{N}}$ the message sent by R_{OT} on input $(1^n, b)$.

Let p, q be primes such that $p = 2q + 1$ and $|p| = n$. Then, we require the OT protocol to satisfy the following requirement. There exists a randomized PPT algorithm R_{OT}^{pub} such that for every $n \in \mathbb{N}$, every $b \in \{0, 1\}$, and every safe prime $p = 2q + 1$, the following two conditions hold:

1. $R_{\text{OT}}(1^n, 0) \overset{\text{c}}{\equiv} R_{\text{OT}}^{pub}(1^n, p)$

2. The output of $R_{\text{OT}}^{pub}(1^n, p)$ consists of components $\{\alpha_i\}_{i=1}^{\text{poly}(n)}$ such that every α_i is a uniform and independent element in an order q subgroup of $\mathbb{Z}_p^*$.

We can formulate the second requirement by simply requiring the output to contain independent and random bits. The difficulty is that we do not know any OT protocol that would satisfy such a requirement. We therefore choose the above formulation. Note that without loss of generality, we can assume that uniform and independent elements can be provided as part of the random tape.[4] We will call algorithm R_{OT}^{pub} the "fake" receiver algorithm.

Concrete instantiation: The existence of R_{OT}^{pub} is extremely crucial for our construction. Unfortunately, no OT protocol satisfying this requirement are known to exist based on *general* assumptions. However, two round OT protocols of [NP01, AIR01] based on the DDH assumption, do satisfy both of our requirements. For concreteness, we fix the Naor-Pinkas oblivious transfer (protocol 4.1 in [NP01]) as our choice, and denote it by OT_{NP}. Algorithm R_{OT}^{pub} in this protocol consists of sending random and independent elements in order q subgroup of $\mathbb{Z}_p^*$. When multiple secrets must be exchanged we simply repeat this protocol in parallel.

Security of OT. In terms of security, the protocols in [NP01, AIR01] are secure against malicious adversaries. However, they do not satisfy the usual simulation based (i.e., "ideal/real") security. Instead, they satisfy the following (informally stated) security notions:

1. *Indistinguishability for receiver:* it ensures that $\{R_{\text{OT}}(1^n, 0)\}_{n \in \mathbb{N}} \overset{\text{c}}{\equiv} \{R_{\text{OT}}(1^n, 1)\}_{n \in \mathbb{N}}$, where $\{R_{\text{OT}}(1^n, b)\}_{n \in \mathbb{N}}$ denotes the message sent by honest receiver on input $(1^n, b)$.

2. *Statistical secrecy for sender:* it ensures either $\{S_{\text{OT}}(1^n, m_0, m_1, q)\}_{n \in \mathbb{N}} \overset{\text{s}}{\equiv} \{S_{\text{OT}}(1^n, m_0, m')\}_{n \in \mathbb{N}}$ or $\{S_{\text{OT}}(1^n, m_0, m_1, q)\}_{n \in \mathbb{N}} \overset{\text{s}}{\equiv} \{S_{\text{OT}}(1^n, m', m_1)\}_{n \in \mathbb{N}}$, where m' is an arbitrary message and $S_{\text{OT}}(1^n, m_0, m_1, q)$ denotes the message sent by the honest sender on input $(1^n, m_0, m_1)$ when receiver's message is q.

[4]This assumption is easily removed by requiring an invertible sampling algorithm for R_{OT}^{pub}, which are known to exist. Also, the two-round requirement is not essential and can be relaxed.

This type of security notion is sufficient for our purpose. A formal description, following [HK12], is given in appendix A.1.

3.2 Extractable Commitments

We will need a perfectly-binding commitment scheme which satisfies the following two properties. First, if a cheating committer C^* successfully completes the protocol, then there exists an extractor algorithm E which outputs the value committed by C^* during the commit stage. Second, there exists a public-coin algorithm C_{pub} such that no cheating receiver can tell if it is interacting with C_{pub} or the honest committing algorithm C. Algorithm C_{pub} is essentially the "fake" committing algorithm for C (much like the fake receiver R^{p}_{OT} define above). Let us first define these properties.

Commit-with-extract. We will actually need a slightly property than mere extraction, called *commit-with-extract* [BL04, Lin01]. Informally, commit-with-extract requires that for every cheating C^*, there exists an extractor E which first simulates the view of the cheating *committer* in an execution with honest receiver; further, if the view is accepting then it also outputs the value committed to in this view. Our specific use requires that the quality of simulation be *statistical.*

Definition 3 (Commit-with-extract [BL04])**.** *Let $n \in \mathbb{N}$ be the security parameter. A perfectly-binding commitment scheme $\Pi_{com} := \langle \mathsf{C}, \mathsf{R} \rangle$ is a* commit-with-extract *scheme if the following holds: there exists a strict* PPT *commitment-extractor E such that for every* PPT *committer C^*, for every $m \in \{0,1\}^n$, every (advice) $z \in \{0,1\}^*$ and every $r \in \{0,1\}^*$, upon input (C^*, m, z, r), machine E outputs a pair, denoted $(E_1(\mathsf{C}^*, m, z, r), E_2(\mathsf{C}^*, m, z, r))$, such that the following conditions hold:*

1. $E_1(\mathsf{C}^*, m, z, r) \overset{\mathrm{s}}{\equiv} \mathrm{VIEW}_{\mathsf{C}^*}[\mathsf{C}^*(m, z; r) \leftrightarrow \mathsf{R}()]$
2. $\Pr[E_2(\mathsf{C}^*, m, z, r)] = \mathit{value}(E_1(\mathsf{C}^*, m, z, r)) \geq 1 - \mathrm{negl}(n)$

where $\mathsf{value}(\cdot)$ *is a deterministic function which outputs either the* unique *value committed to in the view $E_1(\mathsf{C}^*, m, z, r)$, or $\perp$ if no such value exists.*

We say that a perfectly binding commitment scheme Π_{com} admits *public decommitment* if there exists a deterministic polynomial time algorithm D_{com} which on input the *public transcript* of interaction $\widehat{m}$, and the decommitment information d, outputs the *unique* value m committed in $\widehat{m}$. If there is no such value, the algorithm outputs $\perp$. For perfectly binding commitment schemes, the function value is well defined on the public transcripts as well. Therefore, we can write $D_{com}(d, \widehat{m}) = \mathsf{value}(\widehat{m})$.

We now specify our "fake" public-coin sender requirement. Since we are working with DDH based construction, we will use a safe prime $p = 2q + 1$ of length n, (as used in R^{pub}_{OT}).

Let $n \in \mathbb{N}$ be the security parameter. We say that a perfectly binding commitment scheme $\Pi_{com} := \langle \mathsf{C}, \mathsf{R} \rangle$ has a *fake public-coin sender* if there exists an algorithm C_{pub} such that for every malicious PPT R^*, every $m \in \{0,1\}^n$, every safe prime p of length n, every advice $z \in \{0,1\}^*$, the following two conditions hold:

1. $\mathrm{VIEW}_{\mathsf{R}^*}[\mathsf{C}(m) \leftrightarrow \mathsf{R}^*(z)] \overset{\mathrm{c}}{\equiv} \mathrm{VIEW}_{\mathsf{R}^*}[\mathsf{C}_{pub}(p) \leftrightarrow \mathsf{R}^*(z)]$
2. The output of $\mathsf{C}_{pub}(p)$ consists of components $\{\alpha_i\}_{i=1}^{\mathrm{poly}(n)}$ such that for every i: α_i is a uniform and independent element either in $\{0,1\}$ or in an order q subgroup of $\mathbb{Z}_p^*$.

Concrete instantiation. Unfortunately, no commitment protocol satisfying these requirements is known. The central reason behind this is that the fake public-coin sender C_{pub} requirement interferes with the commit-with-extract requirement. In [BL04], Barak and Lindell constructed a commitment protocol with the goal of *strict polynomial time* extraction. We observe that somewhat surprisingly, with some very minor changes, this protocol actually satisfies all our requirements. In particular, this commitment scheme is a *commit-with-extract* scheme, has a *fake public-coin sender*, and admits *public decommitment*. However, as with the OT protocol, this change requires us to use ElGamal [Gam84] and hence DDH (instead of a general trapdoor permutation). For completeness, we present the protocol of [BL04] and explain the required modifications in appendix A.2.

Important notation. For concreteness, fix $\Pi_{com} := \langle \mathsf{C}, \mathsf{R} \rangle$ to be a specific commitment protocol satisfying all three conditions above, and let D_{com} denote it's public decommitment algorithm. Let $\mathbf{L}_{com} := \{(m, \widehat{m}) : \exists d \text{ s.t. } D_{com}(\widehat{m}, d) = m\}$. That is, $\mathbf{L}_{com}$ is an $\mathcal{NP}$-language containing statements $(m, \widehat{m})$ such that $\widehat{m}$ is a commitment-transcript for value m. Let $\mathbf{R}_{com}$ be the corresponding $\mathcal{NP}$-relation so that $\mathbf{R}_{com}((m, \widehat{m}), d) = 1$ if $D_{com}(\widehat{m}, d) = m$ and 0 otherwise.

3.3 Barak's Preamble

In this section, we will recall Barak's non-black-box simulation method. In addition, we will make a slight change to this protocol which requires us to reprove some of the claims. We start by recalling Barak's relation for the complexity class $\mathbf{NTIME}(n^{\log\log(n)})$.

Barak's relation. Let $n \in \mathbb{N}$ be the security parameter, and $\{\mathcal{H}_n\}_n$ be a family of crhf, $h : \{0,1\}^* \to \{0,1\}^n$. Since we are using Naor's commitment scheme, we will have an extra string τ for the commitment scheme sbcom. Barak's relation, $\mathbf{R}_{\mathrm{B}}$ takes as input an instance of the form $\langle h, \tau, c, r \rangle \in \{0,1\}^n \times \{0,1\}^{3n} \times \{0,1\}^{3n^2} \times \{0,1\}^{n+n^2}$ and a witness of the form $\langle M, y, s \rangle \in \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^{\mathrm{poly}(n)}$.

> **Relation:** $\mathbf{R}_{\mathrm{B}}(\langle h, \tau, c, r \rangle, \langle M, y, s \rangle) = 1$ if and only if:
> 1. $|y| \leq |r| - n$.
> 2. $c = \mathsf{sbcom}_\tau(h(M); s)$.
> 3. $M(c, y) = r$ within $n^{\log\log n}$ steps.

Let $\mathbf{L}_{\mathrm{B}}$ be the language corresponding to $\mathbf{R}_{\mathrm{B}}$. We use this more complex version involving y, since it will allow us to successfully simulate even in the presence of leakage queries, which a cheating verifier obtains during the protocol execution.[5]

Universal arguments and statement generation. Universal arguments (UARG) are four-round public-coin interactive argument systems [Kil92, Mic94, BG02], which can be used to prove statements in $\mathbf{L}_{\mathrm{B}}$. Let $\langle P_{\mathrm{UA}}, V_{\mathrm{UA}} \rangle$ be such a system. We will denote the four rounds of this UARG by $\langle \alpha, \beta, \gamma, \delta \rangle$. Consider the following protocol between a party P_{B} and a party V_{B}.

> **Protocol** GenStat: Let $\{\mathcal{H}_n\}_n$ be a family of crhf functions.
> 1. V_{B} sends $h \leftarrow \mathcal{H}_n$ and $\tau \leftarrow \{0,1\}^{3n}$
> 2. P_{B} sends $c \leftarrow \{0,1\}^{3n^2}$

[5]This relation is identical to the one used for constructing bounded concurrent zero-knowledge in constant rounds in [Bar01].

3. V_{B} sends $r \leftarrow \{0,1\}^{n+n^2}$.

Note that that length of r is n^2+n which allows y to be of length at most n^2. Length of c is $3n^2$ since it is supposed to be a commitment to n bits. We have the following lemma:[6]

Composed protocol $\langle P^{\otimes}, V^{\otimes}\rangle$. We define this for convenience. The composed protocol is simply the GenStat protocol followed by an universal argument that the transcript $\sigma := \langle h, \tau, c, r\rangle$ is in $\mathbf{R}_{\text{B}}$. More precisely, strategy $P^{\otimes} := P_{\text{B}} \odot P_{\text{UA}}$ is the composed prover, and $V^{\otimes} := V_{\text{B}} \odot V_{\text{UA}}$ is the composed verifier, where $A \odot B$ denotes the process of running ITM A first, and then continuing ITM B from then onwards.[7] The following lemma states that the composed verifier $V^{\otimes}$ almost always rejects in an interaction with any PPT prover (i.e., it always rejects that $\sigma \in \mathbf{L}_{\text{B}}$).

Lemma 1 ([Bar01])**.** *Suppose that $\{\mathcal{H}_n\}_n$ is a family of* crhf *functions. There exists a negligible function* negl *such that for every* PPT *strategy P^*, every $z \in \{0,1\}^*$, every $r \in \{0,1\}^*$, and every sufficiently large n,*

$$\Pr\left[\text{OUT}_{V^{\otimes}}[P^*(z;r) \leftrightarrow V^{\otimes}()]\right] \leq \text{negl}(n)$$

where the probability is taken over the randomness of $V^{\otimes}$.

The "encrypted" version. In Barak's protocol, an "encrypted" version of the above protocol is used in which the honest prover sends commitments to its UARG-messages (instead of the messages themselves). This is possible to do since the verifier is public coin.

We will use our commit-with-extract scheme $\Pi_{com} := \langle \mathsf{C}, \mathsf{R}\rangle$ for this purpose.[8] Recall that for Π_{com}, there exists a fake public-coin sender algorithm C_{pub} whose execution is indistinguishable from that of C. During the commitment phase, our prover algorithm will follow instructions of C_{pub}; the verifier will continue to use the normal receiver strategy R.

"Encrypted" preamble. $\langle \widehat{P}_{\text{B}}, \widehat{V}_{\text{B}}\rangle$: Let $\{\mathcal{H}_n\}_n$ be a family of crhf functions.

1. $\widehat{P}_{\text{B}}$ and $\widehat{V}_{\text{B}}$ run the GenStat protocol.
   Let $\langle h, \tau, c, r\rangle$ denote the resulting statement.
2. $\widehat{P}_{\text{B}}$ and $\widehat{V}_{\text{B}}$ execute UARG for the statement $\langle h, \tau, c, r\rangle$.
   (a) $\widehat{V}_{\text{B}}$ sends α, obtained from V_{UA}.
   (b) $\widehat{P}_{\text{B}}$ runs C_{pub}, and $\widehat{V}_{\text{B}}$ runs R;
       Let $\widehat{\beta}$ be the commitment transcript.
   (c) $\widehat{V}_{\text{B}}$ sends γ, obtained from V_{UA}.
   (d) $\widehat{P}_{\text{B}}$ runs C_{pub}, and $\widehat{V}_{\text{B}}$ runs R;
       Let $\widehat{\delta}$ be the commitment transcript.

The full transcript of the preamble is $\langle h, \tau, c, r, \alpha, \widehat{\beta}, \gamma, \widehat{\delta}\rangle$.

Since the prover messages are committed, we cannot make a claim along the lines of lemma 1. Therefore, we define the following $\mathcal{NP}$-relation $\mathbf{R}_{sim}$ and claim that it is a "hard" relation. This

[6]The version of Barak's relation that we use is actually a somewhat simplified form of the relation given in [BG02], which results only in a reduction to hash functions that are crhf against circuits of size $n^{\log n}$. By using the more complex version of [BG02], we get a reduction to standard crhf, without affecting any of our claims.

[7]A and B do not share states and run with their own independent inputs.

[8]Recall that Π_{com} is perfectly-binding commitment scheme which satisfies the commit-with-extract notion. In addition, the protocol has a public decommitment algorithm D_{com}, an associated $\mathcal{NP}$-relation $\mathbf{R}_{com}$, and $\mathcal{NP}$-language $\mathbf{L}_{com}$, and a fake public-coin sender algorithm. See section 3.2.

relation simply tests that there exist valid decommitments (d_1, d_2) for strings $\widehat{\beta}, \widehat{\delta}$ so that the transcript is accepted by the UARG verifier.

> **Relation:** $\mathbf{R}_{sim}(\langle h, \tau, c, r, \alpha, \widehat{\beta}, \gamma, \widehat{\delta}\rangle, \langle \beta, d_1, \delta, d_2\rangle) = 1$ if and only if:
> 1. $\mathbf{R}_{com}(\langle \beta, \widehat{\beta}\rangle, d_1) = 1$.
> 2. $\mathbf{R}_{com}(\langle \delta, \widehat{\delta}\rangle, d_2) = 1$.
> 3. $V_{\text{UA}}(h, \tau, c, r, \alpha, \beta, \gamma, \delta) = 1$.

The language corresponding to relation $\mathbf{R}_{sim}$ is denoted by $\mathbf{L}_{sim}$. Also note that $\widehat{P}_{\text{B}}$ sends either random strings of uniform elements in a prime order group of $\mathbb{Z}_p^*$. We have the following lemma.

Lemma 2. *Suppose that $\{\mathcal{H}_n\}_n$ is a family of* crhf *functions. There exists a negligible function* negl *such that for every* PPT *strategy P^*, every $z \in \{0,1\}^*$, every $r \in \{0,1\}^*$, and every sufficiently large n,*

$$\Pr\left[\sigma \leftarrow \text{TRANS}[P^*(z;r) \leftrightarrow \widehat{V}_{\text{B}}(); \sigma \in \mathbf{L}_{sim}]\right] \leq \text{negl}(n)$$

where the probability is taken over the randomness of $\widehat{V}_{\text{B}}$.

Proof. The proof follows almost immediately from the (statistical) commit-with-extract property. Suppose that there exists a machine P^* contradicting the lemma. Then, we construct a machine P^{**} to contradict lemma 1.

Consider the following machine P^{**} which incorporates P^*. Machine P^{**} interacts with an external composed verifier $V^{\otimes}$, forwarding its message to the internal machine P^* up to the point where P^* is about to commit its first UARG message. At this point, P^{**} records the state st_1 of P^*, feeds it to the extractor $E = (E_1, E_2)$ of the commitment scheme. Let the output of the extractor be a simulated state st_1' for P^* (which corresponds to the prover-state at the end of step 2(b) of the protocol). In addition, E also outputs a value β. If $\beta \neq \perp$, P^{**} sends β to $V^{\otimes}$ and receives a message γ. It feeds γ to internal P^* (who is now running from the simulated state st_1'), and repeats the same extraction procedure for the last step, to obtain simulated state st_2' and value δ. If $\delta \neq \perp$, this message is sent to $V^{\otimes}$.

From the statistical simulation property, it follows that the distribution of the final state st_2' of P^* is statistically close to its view in a real-execution with an honest $\widehat{V}_{\text{B}}$. As a result, if $\widetilde{\sigma}$ denotes the transcript of interaction (contained in the state st_2'), $\Pr[\widetilde{\sigma} \in \mathbf{L}_{sim}]$ is close to the probability in the lemma. Further, by the correctness of extraction, the extracted values β, δ are indeed correct openings except with negligible probability. It therefore follows that P^* convinces $V^{\otimes}$ with noticeable probability. This contradicts lemma 1. ■

Remark. We wish to remak that the above proof relies crucially on the *statistical* simulation guarantee of E_1 (in simulating the view of the committer, see definition 3). This is because the statement $\sigma \notin \mathbf{L}_{sim}$ is not efficiently verifiable, and therefore mere computational indistinguishability will not lead to any contradiction.

4 Conditional Disclosure via Garbled Circuits

Yao's garbled circuit method [Yao86] allows two parties to compute any arbitrary function f of their inputs in a "secure" manner. Without loss of generality, let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.

The method. The garbled circuit method specifies two polynomial time algorithms ($\mathsf{Garble}, \mathsf{Eval}$). Algorithm Garble is randomized; on input 1^n and the description of a circuit (that computes) f, it outputs a triplet $(\mathcal{C}, \mathsf{key}_0, \mathsf{key}_1)$. $\mathcal{C}$, which consists of a set of tables containing encrypted values, is called the *garbled circuit*; and $\mathsf{key}_0 = \{(k^0_{0,i}, k^1_{0,i})\}_{i=1}^n$ and $\mathsf{key}_1 = \{(k^0_{1,i}), k^1_{1,i}\}_{i=1}^n$ are called the keys. Let $a = a_1, \ldots, a_n$ and $b = b_1, \ldots, b_n$ be binary strings. Algorithm Eval, on input $(\mathcal{C}, K_a, K_b)$ outputs a value $v \in \{0,1\}^n$ such that if $K_a = \{k^{a_i}_{0,i}\}$ and $K_b = \{k^{b_i}_{1,i}\}$ then $v = f(a, b)$.

For an $\mathcal{NP}$-relation R, and $\sigma \in \{0,1\}^*$, let $f_{\sigma,R}$ be the following function.

Function $f_{\sigma,R}(\omega, m)$:
If $R(\sigma, \omega) = 1$, `output` m; otherwise `output` $0^{|m|}$.

That is, $f_{\sigma,R}$ discloses m if and only if ω is a valid witness for the statement σ. We will use the garbled circuit method for such functions $f_{\sigma,R}$. Jumping ahead, we will use $f_{\sigma,\mathbf{R}_{sim}}$ for the $\mathcal{NP}$-relation $\mathbf{R}_{sim}$ described in section 3.3.

Conditional disclosure via garbled-circuits. In the two party setting, one party prepares the garbled circuit $\mathcal{C}$ and sends the keys K_b corresponding to her input b to the other party. An OT protocol is used by the first party to receive keys K_a for her input a, so that it can execute the evaluation algorithm. This allows the receiver of the garbled circuit (and OT) to learn $f(a, b)$ but "nothing more". In addition, receiver's input remains secure due to OT-security for receiver.

Looking forward, we will require our protocol so that it will admit a "fake" receiver algorithm. Therefore, we will use the Naor-Pinkas OT protocol, denoted OT_{NP} (see section 3.1). For a technical reason, our protocol starts by first executing steps of OT_{NP}, and then executes the garbled circuit step. Note that the first step involves n parallel executions of OT, one for each input bit. The resulting two-round conditional disclosure protocol, Π_{cd}, is as follows.

Protocol Π_{cd} for computing $f_{\sigma,R}(\omega, m)$: The protocol consists of two parties, a receiver R_{cd} and a sender S_{cd}. R_{cd}'s private input is bit string $\omega = \omega_1, \ldots, \omega_n$, and S_{cd}'s private input is bit string $m = m_1, \ldots, m_n$. The common input to the parties is the description of the function $f_{\sigma,R}$ as a circuit (equivalently, just σ).

1. R_{cd} computes $v = (v_1, \ldots, v_n)$, where v_i is the first message of OT_{NP} using the input ω_i and fresh randomness for $i \in [n]$. It then sends v.
2. S_{cd} prepares a garbled circuit for the function $f_{\sigma,R}$: $(\mathcal{C}_{\sigma,R}, \mathsf{key}_0, \mathsf{key}_1) \leftarrow \mathsf{Garble}(f_{\sigma,R})$. Next, S_{cd} prepares $v' = (v'_1, \ldots, v'_n)$ where v'_i is the second message of OT_{NP} computed using $(k^0_{0,i}, k^1_{0,i})$ as sender's input and v_i as receiver's first message. Here the keys $(k^0_{0,i}, k^1_{0,i})$ are the i^{th} component of key_0. Finally, let K_m denote the keys taken from key_1 corresponding to m. S_{cd} sends $(\mathcal{C}_{\sigma,R}, v', K_m)$.

Recall that OT_{NP} is a two-round protocol, it provides statistical secrecy for the sender, and has a fake public-coin receiver. Also recall that OT_{NP} does not satisfy the the standard simulation-based security. As a result, we cannot directly use known results about the security of Yao's protocol. Nevertheless, we can make weaker indistinguishability-style claims which suffice for our purpose. First notice that the OT-security for receiver, intuitively guarantees indistinguishability for the input of R_{cd}. For the sender, we can prove the following claim.

Lemma 3 (Security for sender)**.** *Let $L \in \mathcal{NP}$ with witness relation R and $\sigma \in \{0,1\}^*$. For the security parameter n, let $S_{cd}(1^n, f_{\sigma,R}, m, q)$ represent the response of the honest sender (of protocol*

Π_{cd}), with input $(f_{\sigma,R}, m)$ when receiver's first message is q. Then, for every pair of distinct messages (m, m'), every $q \in \{0,1\}^*$ (from a possibly malicious PPT *receiver), and* <u>*every $\sigma \notin L$*</u>*, it holds that $S_{cd}(1^n, f_{\sigma,R}, m, q) \overset{c}{\equiv} S_{cd}(^n, f_{\sigma,R}, m', q)$.*

Proof. First, observe that when $\sigma \notin L$, $f_{\sigma,R}$ is a constant function which always outputs an all-zero string irrespective of inputs. Let (ω, m) and (ω', m') be two distinct set of inputs such that $f_{\sigma,R}(\omega, m) = f_{\sigma,R}(\omega', m')$. For such pairs of inputs, it follows from the security of Yao's construction that the following two distributions are computationally indistinguishable (see, e.g., [LP09] for details):

$$(\mathcal{C}, K_\omega, K_m) \overset{c}{\equiv} (\mathcal{C}, K_{\omega'}, K_{m'}) \tag{2}$$

We show that if the lemma is not true, then there exists a PPT adversary contradicting (2).

Suppose that the lemma is not true. This means, that there exists a PPT adversary R^*_{cd}, a security parameter n, a statement $\sigma_n \notin L$, a first message q_n, and a pair of messages (m, m') such that adversary's advantage in distinguishing $S_{cd}(1^n, f_{\sigma_n,R}, m, q_n)$ from $S_{cd}(1^n, f_{\sigma_n,R}, m', q_n)$ is noticeable. Fix such a value of n, m, m', σ_n, q_n out of infinitely many.

Recall that q_n is the first message of OT_{NP}. There exists at most one valid input to the honest receiver algorithm of OT_{NP} which results in q_n as its first message. Let this input be denoted by ω^*_n.[9] Without loss of generality, R^*_{cd} is deterministic, and therefore q_n is always fixed for this particular value of n and σ_n. Therefore, ω^*_n is also fixed for this n, and can be provided *non-uniformly* to a distinguishing machine.

We now construct an adversary R^*_{gc} for the garbled-circuit algorithm to violate (2). R^*_{gc} incorporates R^*_{cd}, and receives ω^*_n as its non-uniform advice. It then starts an internal execution of Π_{cd} with R^*_{cd} receiving q_n as the first message. When this happens, $AR*_{gc}$ sends ω^*_n to an outside party. The outside party constructs a garbled circuit $\mathcal{C}_{\sigma_n,R}$ for $f_{\sigma_n,R}$, and keys $K_{\omega^*_n}$ and K_{ch} for inputs ω^*_n and ch (which will be either to set to m or m'). The triplet $(\mathcal{C}_{\sigma_n,R}, K_{\omega^*_n}, K_{ch})$ is given to R^*_{gc}.

At this point, R^*_{gc} constructs key'_0 so that if keys were taken from key_0 for the input ω^*_n, they will result in the vector $K_{\omega^*_n}$; this is done by using keys from the vector $K_{\omega^*_n}$ and filling in randomly generated keys in appropriate locations of key'_0. To complete the internal simulation, R^*_{gc} now prepares the second message v_2 of OT_{NP} by using pairs of inputs from key'_0 and q_n as receiver's first message. It then gives $(\mathcal{C}_{\sigma_n,R}, v_2, K_{ch})$ to R^*_{cd}, and outputs whatever R^*_{cd} outputs.

Since OT_{NP} guarantees statistical secrecy for the sender, the view of R^*_{cd} in the internal simulation is *statistically* close to it's view in the real execution of Π_{cd} with sender's input being ch. Observe that the view of R^*_{gc} either consists of $(\mathcal{C}_{\sigma_n,R}, K_{\omega^*_n}, K_m)$ or $(\mathcal{C}_{\sigma_n,R}, K_{\omega^*_n}, K_{m'})$ depending upon whether $ch = m$ or $ch = m'$. It follow that the distinguishing advantage of R^*_{gc} in violating (2) is negligibly close to that of R^*_{cd} in distinguishing the distributions in the lemma. ■

5 A Constant Round Protocol

In this section we will present our constant round protocol. The protocol will use the dual simulation idea, introduced in [GJS11], as an important tool. To simplify the exposition and the proofs, we isolate a part of the protocol from [GJS11], and present it as a separate building block.[10]

[9]If some, or all, bits of ω^*_n are not uniquely defined, then the undefined bits are set to 0.

[10]The only difference is that the challenge-response slots in the [GJS11] protocol have been removed. As a result, many other parameters of their protocol become irrelevant, and also do not appear in this protocol. This does not affect the soundness of the protocol.

Shortened GJS protocol $\langle P_{\text{GJS}}, V_{\text{GJS}} \rangle$. The common input is an n vertex graph G in the form of an adjacency matrix, and prover's auxiliary input is a Hamiltonian cycle H in G. The protocol proceeds in following three steps.

1. Commitment stage:
   - (a) P_{GJS} sends a random string ρ.
   - (b) V_{GJS} sends $\widehat{t_1} = \mathsf{shcom}_\rho(t_1; s_1)$ and $\widehat{ch} = \mathsf{shcom}_\rho(ch; s_2)$, where $t_1 \leftarrow \{0,1\}^{3n^4}$, $ch \in \{0,1\}^n$, and $s_1, s_2 \leftarrow \{0,1\}^{\text{poly}(n)}$.
2. Coin flipping stage:
   - (a) P_{GJS} sends a random string t_2.
   - (b) V_{GJS} opens $\widehat{t_1}$ by sending (t_1, s_1). Let $\mathbf{t} = t_1 \oplus t_2$.
3. Blum Hamiltonicity protocol:
   - (a) Let $\mathbf{t} = \mathbf{t}_1, \ldots, \mathbf{t}_{n^3}$ so that $|\mathbf{t}_i| = 3n$ for $i \in [n^3]$. Prover chooses n random permutations $\pi_1, \ldots, \pi_n$ and sets $G_i = \pi_i(G)$ for each $i \in [n]$. It then commits to each bit b_j in G_i using $\mathsf{sbcom}_{\mathbf{t}_{i \times j}}$.
   - (b) Verifier opens to $\widehat{ch}$ by sending (ch, s_2).
   - (c) Let $ch = ch_1, \ldots, ch_n$. For every $i \in [n]$, if $ch_i = 0$ then prover opens each edge in G_i and reveals π_i; else, it opens edges of the cycle in G_i.

The following lemma has been shown in [GJS11].

Lemma 4 ([GJS11])**.** *Protocol $\langle P_{\text{GJS}}, V_{\text{GJS}} \rangle$ is a sound interactive argument system for all of $\mathcal{NP}$.*

5.1 Our Protocol

We are now ready to present our protocol $\langle P, V \rangle$. The protocol starts with an execution of the "encrypted" preamble protocol (see section 3.3); this is followed by the first i.e., commitment, stage of the GJS protocol. Before completing the GJS protocol, verifier executes the garbled-circuit protocol Π_{cd} for $f_{\sigma, \mathbf{R}_{sim}}$ and a specific m (described shortly), and proves using an szkaok that this step was performed honestly. This will enable the simulator to extract useful information in m. Finally, the rest of the GJS protocol is executed to complete the proof. The full description of the protocol is given below.

Protocol $\langle P, V\rangle$. The common input consists of 1^n, and an n vertex graph G in the form of its adjacency matrix. Prover's private input is a Hamiltonian cycle H in G.

1. **"Encrypted" preamble:** $P \Rightarrow V$

   P and V run Barak's encrypted preamble. P runs the public-coin strategy $\widehat{P}_{\mathrm{B}}$, and V runs strategy $\widehat{V}_{\mathrm{B}}$. Let the transcript be $\sigma := \langle h, \tau, c, r, \alpha, \widehat{\beta}, \gamma, \widehat{\delta}\rangle$.

2. **Commitment step:** $V \Rightarrow P$

   P and V run the first, i.e. commitment, step of $\langle P_{\mathrm{GJS}}, V_{\mathrm{GJS}}\rangle$.

   (a) P sends a random string ρ

   (b) V sends $\widehat{t_1} = \mathsf{shcom}_\rho(t_1; s_1)$ and $\widehat{ch} = \mathsf{shcom}_\rho(ch; s_2)$, where $t_1 \leftarrow \{0,1\}^{3n^4}$, $ch \leftarrow \{0,1\}^n$, and $s_1, s_2 \leftarrow \{0,1\}^{\mathrm{poly}(n)}$; let $m := (t_1, s_1, ch, s_2)$.

3. **Garbled-circuit step:** $V \Rightarrow P$

   P and V run the *two-round* garbled circuit protocol, Π_{cd}, for the function $f_{\sigma,\mathbf{R}_{sim}}$. V acts as the sender with private input m.

   (a) P runs the fake receiver, $v_1 \leftarrow R^{pub}_{\mathrm{OT}}(1^n, p)$ for a random safe prime p; sends v_1.

   (b) V sends $(\mathcal{C}, v_2, K_m) \leftarrow S_{cd}(f_{\sigma,R}, m, v_1; s_3)$, using fresh coins s_3.

4. **Proof of correctness:** $V \Rightarrow P$

   V proves to P using public-coin szkaok Π_{PR} the knowledge of s_3 and $m = (t_1, s_1, ch, s_2)$ so that:

   (a) $\widehat{t_1} = \mathsf{shcom}_\rho(t_1; s_1)$,

   (b) $\widehat{ch} = \mathsf{shcom}_\rho(ch; s_2)$,

   (c) $S_{cd}(f_{\sigma,R}, m, v_1; s_3) = (\mathcal{C}, v_2, K_m)$.

5. **Final step:** $P \Rightarrow V$

   P and V complete all remaining five rounds of $\langle P_{\mathrm{GJS}}, V_{\mathrm{GJS}}\rangle$. P uses H as the witness.

It is easy to see that our protocol has constant rounds. The completeness of the protocol follows directly from the completeness of $\langle P_{\mathrm{GJS}}, V_{\mathrm{GJS}}\rangle$. In next two sections, we prove the soundness and zero-knowledge of this protocol. Note that the the prover is actually "public coin" *up until the final step.* The proof of theorem 1 follows from the proof of soundness (section 5.2) and leakage-resilient zero-knowledge (section 5.3).

5.2 Proving Soundness

We prove the soundness of our protocol by reducing it to the soundness of GJS protocol (see lemma 4, section 5.1). To do so, we proceed in three steps. First, from lemma 2, it should hold that if a cheating prover P^* succeeds with noticeable probability, then it must do so even if the transcript σ of the preamble is not in $\mathbf{L}_{sim}$. In the next step, we construct new machine $P{*}_1$ from P^* which interacts with a modified verifier V_1. The verifier V_1 uses the statistical simulator of szkaok (in the garbled circuit-step) instead of using the prover. P^* should convince V_1 with noticeable probability as well. Finally, V_1 replaces the value m in the garbled circuit by $0^{|m|}$, as its next step. We argue that doing so also maintains noticeable probability of success for P_1^*. This is since when $\sigma \notin \mathbf{L}_{sim}$, changing m to $0^{|m|}$ maintains indistinguishability from lemma 3. At this point, V_1 is essentially "decoupled" and can receive GJS-messages from an external GJS-verifier. The full proof appears below.

Lemma 5. *Protocol $\langle P, V\rangle$ is sound.*

Proof. We will show that if our protocol is not sound, then the GJS protocol is also not sound, contradicting lemma 4.

Suppose that there exists a PPT machine P^* which violates the soundness of our protocol for infinitely many n. Fix one such security parameter n and a non-Hamiltonian graph G_n, such that P^* succeeds in convincing an honest V with probability $\epsilon_n \geq 1/p(n)$ for some polynomial $p(n)$.

Consider the state of prover P^* at the conclusion of the preamble. Suppose that the transcript of the preamble is σ. Let P^*_σ denote this residual prover strategy. Let V_1 denote the residual honest verifier strategy.[11] We have the following (via a standard averaging argument, see appendix B).

Claim 1. *There exist at least an $\epsilon_n/4$ fraction of prefix strings σ such that: (a) $\sigma \notin \mathbf{L}_{sim}$, and (b) P^*_σ successfully proves G_n to V_1 with probability at least $\epsilon_n/2$.*

Fix one such prefix $\sigma \notin \mathbf{L}_{sim}$ and the corresponding prover strategy P^*_σ. Now we consider the following verifier V_2. Verifier V_2 is identical to V_1 except that it uses the simulator S_{PR} of the szkaok protocol Π_{PR}. We note that since S_{PR} is a non-black-box simulator, it requires the program of the machine whose view it must simulate. Therefore, V_2 also receives the program of P^*_σ as its input. Observe that this is sufficient for successfully executing algorithm V_2.

Let $\epsilon' \geq \epsilon_n/2$ be the success probability of P^*_σ in convincing V_1. From the statistical simulation property of szkaok, it holds that distributions $\text{TRANS}[P^*_\sigma(x,z) \leftrightarrow V_1()]$ and $\text{TRANS}[P^*(x,z) \leftrightarrow V_1(P^*)]$ are statistically close. Therefore, we have that P^*_σ convinces V_2 with probability $\epsilon'' \geq \epsilon' - \text{negl}(n) \geq \epsilon_n/4$.

We now proceed to the final step and consider the following verifier V_3. Verifier V_3 is identical to V_2 except that instead of using $m = (t_1, s_1, ch, s_2)$ it uses $m' = 0^{|m|}$ to send the second message of the garbled-circuit step. Let ϵ''' be the probability that P^*_σ successfully convinces V_3.

Let us now compare the two executions of P^*_σ with V_2 and V_3. In both executions, the first message received by P^*_σ is $u := (\widehat{t}, \widehat{ch})$, and distributed identically. Let $\lambda_2(u)$ denote the probability that P^*_σ successfully convinces V_2 after receiving u; define $\lambda_3(u)$ analogously in case of V_3. Then, denoting by $\Pr[u]$ the probability that u is sent, we have:

$$\epsilon'' = \sum_u \Pr[u] \cdot \lambda_2(u) \quad \text{and} \quad \epsilon''' = \sum_u \Pr[u] \cdot \lambda_3(u)$$

Since $\sigma \notin L$, from lemma 3 we have that $\lambda_2(u)$ and $\lambda_3(u)$ are negligibly close; in particular $\lambda_3(u) \geq \lambda_2(u) - \text{negl}(n)$. Let $\mathcal{G}$ be the set of messages u such that $\lambda_2(u) \geq \epsilon''/2$. This means that for every $u \in \mathcal{G}$, $\lambda_3(u) \geq \epsilon''/2 - \text{negl}(n) \geq \epsilon_n/16$. Therefore,

$$\epsilon''' \geq \sum_{u \in \mathcal{G}} \Pr[u] \cdot \lambda_3(u) + 0 \geq \frac{\epsilon_n}{16} \sum_{u \in \mathcal{G}} \Pr[u]$$

Using a standard averaging argument (along the lines of claim 1), we can show that $\Pr[u \in \mathcal{G}] \geq \epsilon''/2 \geq \epsilon_n/8$. Therefore $\epsilon''' \geq \epsilon_n^2/128$. It is now straightforward to construct a prover P^*_{GJS} for breaking the soundness of GJS protocol.

[11] Observe that the randomness of V in the preamble is not needed any other step. Therefore, such a residual honest verifier is well defined: it simply executes all steps of V from step 2 onwards with fresh randomness.

The machine P^*_{GJS} incorporates the prover P^* that violates the soundness of our protocol. It then interacts with P^* honest to conclude the encrypted preamble of (our) protocol $\langle P, V\rangle$. This results in statement σ; at this point, it freezes the state of P^*, to obtain the strategy P^*_σ. Now P^*_{GJS} starts an interaction between this residual prover P^*_σ and a new machine V'_3.

The machine V'_3 is identical to V_3 except that it routes all messages corresponding to the GJS protocol to an external honest verifier V_{GJS}. To successfully convince V_{GJS} that G_n is a Hamiltonian graph, P^*_{GJS} starts an internal interaction between P^*_σ and V'_3 on common input G_n. When a verifier-message of the GJS protocol is required, the machine asks for this message from the external V_{GJS}. Likewise, when a prover message of the GJS protocol is produced, it is sent to the external V_{GJS}. P^*_{GJS} halts when the entire internal execution halts.

It is easy to see that the interaction of P^*_{GJS} with V_{GJS} is identical to that of P^*_σ with V_3. Therefore, P^*_{GJS} has convincing probability $\frac{\epsilon_n}{4} \times \frac{\epsilon_n^2}{128} = O(\epsilon_n^3)$. This contradicts lemma 4 if ϵ_n is not negligible.

■

5.3 Proving leakage-resilient zero-knowledge

We use Barak's non-black-box simulation idea along with GJS simulation. We now quickly highlight the main ideas in the simulation. Let V^* be an arbitrary PPT verifier whose program is given as an input to the simulator S. There are for main ideas:

1. First, the simulation uses V^*'s code to execute the preamble in such a way, that at the end of the preamble, $\sigma \in \mathbf{L}_{sim}$. In addition, the simulator will also have a witness ω so that $\mathbf{R}_{sim}(\sigma, \omega) = 1$. The properties of the components used in the preamble (in particular the use of fake sampling algorithms that are public coin) guarantee that simulator's actions in the preamble are indistinguishable from a real execution with an honest prover. In addition, it is easy to answer leakage queries since the messages exchanged so far represent the entire random-tape of the prover at this point. This allows the simulator to answer leakage queries by simply appending these messages to the state, and sending an appropriate query to the leakage oracle.

2. Next, the simulator will use ω in the garbled circuit step to obtain keys K_ω. Once again, since the first message of $\mathsf{OT}_{\mathrm{NP}}$ provides indistinguishability for receiver's input, this step does not affect the simulation. Further, since P is public coin in this step as well, the simulator can continue to answer leakage queries as before.

3. Having obtained K_ω along with $\mathcal{C}, K_m$ in the garbled circuit step, the simulator can evaluate the $\mathcal{C}$ and learn $f_{\sigma,\mathbf{R}_{sim}}(\omega, m)$ to learn m. By the soundness of szkaok of the next step, it is guaranteed that m contains valid openings (t_1, s_1, ch, s_2) for $\widehat{t_1}$ and $\widehat{ch}$.

4. Finally, observe that (t_1, ch) is precisely the information needed by the GJS simulation method to successfully simulate the last step, while answering leakage queries properly. Briefly, ch is the challenge for Blum's protocol, and a first message can be created by the simulator to successfully answer V^*'s challenge in the last message. At the same time, since t_1 is known prior to the coin-flipping stage of the GJS protocol (see section 5), the simulator will have the ability to equivocate in Naor's commitment scheme, allowing it to successfully answer leakage queries.

An important point to note is that if V^* asks more than n^2 bits of leakage after receiving c and before sending r (see GenStat), the simulator will not be able to ensure that $\sigma \in \mathbf{L}_{sim}$. However, if this happens, the simulator can simply ask for the entire witness H from the leakage oracle since the length of leakage is more than the witness size. The simulator can then continue to run like the honest prover and output a view.

Let $\mathcal{L}_H^n(\cdot)$ denote the leakage oracle parameterized by a Hamiltonian cycle H, and the security parameter n. Our non-black-box simulator S will have access to $\mathcal{L}_H^n(\cdot)$, to answer leakage queries. A leakage query is a function $F(H, \texttt{tape})$ (represented as a circuit), which takes as input the witness H and prover's random tape $\texttt{tape}$, and outputs the leakage bits. The description of the simulator follows.[12]

The simulator S. Let V^* be a PPT cheating verifier. The simulator receives as input the security parameter 1^n, an n vertex Hamiltonian graph G, and the description of V^*. The simulator starts by fixing V^*'s inputs and random tape, to completely fix its program. It then starts interacting with V^* as described below. The simulator aborts if V^* aborts (e.g., by sending invalid messages). Let $\texttt{tape}$ be a string initially empty.

1. **"Encrypted" preamble:**
   (a) Suppose that V^* sends it's first message (h, τ). If V^* makes a leakage query $F(\cdot,\cdot)$, S sends $F(\cdot, \texttt{tape})$ to $\mathcal{L}_H^n$, and feeds the resulting answer to V^*. Let V^{**} denote the current state of the verifier.

   (b) S computes $c = \mathsf{sbcom}_\tau(h(V^{**}); s)$ for a random $s \leftarrow \{0,1\}^{\text{poly}(n)}$, and feeds c to the verifier. It then appends c to $\texttt{tape} := \texttt{tape} \parallel c$.

   <u>*Leakage query:*</u> If V^* sends a leakage function F whose output is *more* than n^2 bits, S asks for the witness H from $\mathcal{L}_H^n$ and answers the query as $F(H, \texttt{tape})$. Otherwise, the query is answered as before; let y be the answered returned by $\mathcal{L}_H^n$, so that $|y| \leq n^2 \leq |r| - n$.

   (c) If V^* responds, let r be the string obtained as its next message.

   If S holds the cycle H, it will continue its action as an honest prover. Otherwise, S holds the "fake" witness $\langle V^{**}, y, s\rangle$ for $\langle h, \tau, c, r\rangle$, to satisfy Barak's relation $\mathbf{R}_{\text{B}}$. In the later case, it proceeds as follows.

   (d) When V^* sends the next message α, S uses P_{UA} with the witness $\langle V^{**}, y, s\rangle$, to obtain a string β.

   (e) S commits β to V^* using the committing algorithm C. Let $\widehat{\beta}$ be the transcript.

   (f) When V^* sends the next message γ, S computes the next UARG message δ.

   (g) S commits δ to V^* using the committing algorithm C. Let $\widehat{\delta}$ be the transcript.

   <u>*Leakage queries:*</u> At the end of each round, S appends its messages to $\texttt{tape}$, and answers (any) leakage function F by sending $F(\cdot, \texttt{tape})$ to $\mathcal{L}_H^n$ and returning the answer.

   <u>*Fake witness:*</u> Let $\sigma := \langle h, \tau, c, r, \alpha, \widehat{\beta}, \gamma, \widehat{\delta}\rangle$. At this point, S holds witness $\omega := \langle \beta, d_1, \delta, d_2\rangle$ such that $\mathbf{R}_{sim}(\sigma, \omega) = 1$.

2. **Commitment step:** S executes the commitment step with V^* honestly: S sends a random string ρ, and receives a pair $(\widehat{t_1}, \widehat{ch})$. Leakage queries are answered as in the previous step.

3. **Garbled-circuit step:** S uses the receiver algorithm R_{cd} of the garbled-circuit protocol; the input to R_{cd} is the (fake) witness ω, and let v_1 be its output. S feeds v_1 to V^*. Let $(\mathcal{C}, v_2, K_m)$ be the response of V^*. Leakage queries are answered as in the previous step.

[12]We will need to describe the actions of S in all steps, including messages in the GenStat protocol, and the GJS protocol. We recommend a quick look on sections 3.3 and 5.

4. **Proof of correctness:** S plays this step honestly, and receives a proof of knowledge of inputs $m = (t_1, s_1, ch, s_2)$ and s_3 so that $\widehat{t_1} = \mathsf{shcom}_\rho(t_1; s_1)$, $\widehat{ch} = \mathsf{shcom}_\rho(ch; s_2)$, and $S_{cd}(f_{\sigma,\mathbf{R}_{sim}}, m, v_1; s_3) = (\mathcal{C}, v_2, K_m)$. Leakage queries are answered as in the previous step. If the proof is accepting, S obtains keys K_ω from the OT messages (v_1, v_2), and evaluates the garbled circuit to learn $f_{\sigma,\mathbf{R}_{sim}}(\omega, m) = m$. If m does not contain valid openings to $(\widehat{t_1}, \widehat{ch})$, S outputs a special symbol $\mathtt{fail}_1$.

5. **Final step:** S now proceeds identically to the GJS-simulator using the trapdoors (t_1, ch) to simulate (and answer leakage queries).[13] For completeness, we recall these steps here:

   (a) *Coin flipping:* S must choose string t_2 in a way so that it can equivocate. Let the input $t_1 = t'_0, \dots, t'_{3k^3-1}$. Then for every $i \in \{0, \dots, 3k^3 - 1\}$, S sets $t''_i = t'_i \oplus \mathsf{prg}(s_i^0) \oplus \mathsf{prg}(s_i^1)$ for randomly chosen seeds (s_i^0, s_i^1). It then sends t_2 which is a concatenation of all t''_i. If V^* responds with a valid opening (t_1^*, s_1^*) of $\widehat{t_1}$ such that $t_1^* \neq t_1$, S aborts and outputs $(\mathtt{fail}_2, t_1^*, s_1^*)$. Otherwise, it sets $\mathtt{t} = t_1 \oplus t_2$. Note that $\mathtt{t}_i = \mathsf{prg}(s_i^0) \oplus \mathsf{prg}(s_i^1)$, and allows for equivocation.[14] Leakage queries are handled as before.

   (b) *Blum's protocol:* S prepares the first message by using $ch = ch_1, \dots, ch_n$ as the challenge of the verifier. For $i \in [n]$, if $ch_i = 0$ it commits to a random $G_i = \pi_i(G)$; otherwise it commits to a random n-cycle graph G_i. If V^* sends a valid opening (ch^*, s_2^*) of $\widehat{ch}$ such that $ch^* \neq ch$, S outputs $(\mathtt{fail}_2, ch^*, s^*)$. Otherwise it answers with correct openings depending on the challenge.

   Leakage queries. To handle leakage queries, a special deterministic function $\mathcal{R}(H, \mathtt{tape})$ (described in [GJS11]) is first defined. This function uses the randomness used by Blum's simulator, and constructs randomness r' such that Blum's prover with witness H and randomness r' will result in the exact same transcript as output by Blum's simulator (in the final step). Description of $\mathcal{R}$ includes the relevant equivocation trapdoors (s_i^0, s_i^1) for Naor's commitment for this purpose; $\mathcal{R}$ outputs $\mathtt{tape} := \mathtt{tape} \parallel r'$. Note that the simulator never runs $\mathcal{R}$ by itself. Upon receiving a leakage query F, S sends the query F' to $\mathcal{L}_H^n$, so that $F'(H) = F(H, \mathcal{R}(H, \mathtt{tape}))$. See [GJS11] (section 3.3.1, page 16, full version) for a complete description of this function.

This completes the description of the simulator. S outputs whatever is the final view of V^*. The bound on the number of leakage bits is seen to hold trivially. We now proceed to show the indistinguishability of simulation.

Indistinguishability of simulation. To prove that the view output by S is computationally indistinguishable from the real view, we first design the following hybrid machines, $M_0, \dots, M_8$. M_0 receives the cycle H as input, whereas M_8 does not.

M_0: This hybrid receives witness H and the program of V^* as input, and interacts with V^* just like the honest prover P. Leakage queries are answered directly based on the witness and prover's random tape.

M_1: This hybrid differs from M_0 only in preparation of the commitment c: it commits to $h(V^{**})$ instead of sending a random string in $\{0,1\}^{3n^2}$; here V^{**} is the state of V^* before receiving c.

[13]We only need simulation steps from commitment phase, which is straight-line given (t_1, ch).

[14]To commit, S sends $\mathsf{prg}(s_i^0)$ as its commitment which can be opened to 0 by sending s_i^0, and 1 by sending s_i^1.

Leakage queries are answered as in M_0, assuming that c is the public randomness of P in this round.

M_2: This hybrid differs from M_1 only in the execution of "encrypted" UARG. While M_1 uses algorithm C_{pub}, M_2 acts as described in steps 1(d) through 1(g) of S. That is, M_2 uses $\langle V^{**}, y, s\rangle$ to compute messages β, δ of the UARG prover, and then commits to them using the honest committing algorithm C. To answer leakage queries, M_2 considers messages sent by algorithm C as if they were prover's public coins in these steps; it then answers the queries as before.

M_2 also records witness $\omega := \langle \beta, d_1, \delta, d_2\rangle$ for transcript $\sigma := \langle h, \tau, c, r, \alpha, \widehat{\beta}, \gamma, \widehat{\delta}\rangle$; here d_1 and d_2 are strings that decommit $\widehat{\beta}$ to β and $\widehat{\delta}$ to δ respectively.

M_3: This hybrid is identical to M_2 in all but the garbled-circuit step. M_3 uses witness ω as the input to the honest receiver algorithm R_{cd} to compute the message v_1 (which in turn is the first message of OT_{NP}). Leakage queries are answered as in previous step.

M_4: This hybrid is same as M_3, except that at the end of step 4, if szkaok is accepting, it evaluates the circuit $\mathcal{C}$ to learn a message m. If m does not contain valid openings to both $(\widehat{t}_1, \widehat{ch})$, M_4 aborts and outputs a special symbol $\mathtt{fail}_1$. Leakage queries are answered as in M_3.

M_5: This hybrid is same as M_4, except that in coin flipping stage M_5 sends a string t_2 such that the final string $\mathbf{t} = t_1 \oplus t_2$ allows for equivocation for each of its component strings (for Naor's commitment sbcom, see step 5(a) of S). Leakage queries are handled as before.

M_6: This hybrid differs from M_5 only in how it reacts to V^*'s opening of $(\widehat{t}_1, \widehat{ch})$. If V^* responds with a valid opening (t_1^*, s_1^*) of $\widehat{t}_1$ (in the coin flipping stage) such that $t_1^* \neq t_1$, M_6 aborts with output $(\mathtt{fail}_2, t_1^*, s_1^*)$. It does the same thing if V^* sends with a valid opening (ch^*, s_2^*) of $\widehat{ch}$ (in Blum's protocol) such that $ch^* \neq ch$. Leakage queries are handled as before.

M_7: This hybrid differs from M_6 only in Blum's protocol. Instead of using H, it uses the simulator of Blum's protocol (with the challenge ch) to successfully complete this step. Leakage queries are handled as follows. M_7 uses a special deterministic function $\mathcal{R}(H, \mathtt{tape})$ (described in [GJS11]) which uses the randomness used by Blum's simulator and constructs randomness r' such that Blum's prover with witness H and randomness r' will result in the exact same transcript as output by Blum's simulator (in final step). Description of $\mathcal{R}$ includes the equivocation trapdoor of Naor's commitment for this purpose; $\mathcal{R}$ outputs $\mathtt{tape} := \mathtt{tape} \parallel r'$. To answer leakage query F, M_7 sends $F(\cdot, \mathtt{tape})$ to $\mathcal{L}_H^n$.

M_8: This hybrid is same as M_7 except that it handles leakage queries slightly differently. Upon receiving F, it constructs leakage query F' satisfying $F'(x, \mathtt{tape}) = F(x, \mathcal{R}(x, \mathtt{tape}))$; and sends $F'(\cdot, \mathtt{tape})$ to $\mathcal{L}_H^n$. Note that M_8 does not require access to H at all, and is identical to our simulator S.

Lemma 6. *Protocol $\langle P, V\rangle$ is a leakage-resilient zero-knowledge protocol.*

Proof. To prove the lemma, we only need to show that for every V^* and every valid (G, H), the output of S is computationally indistinguishable from the view of V^* in an interaction with the real prover $P(G, H)$. To do this, we show that the output of hybrid M_0 is indistinguishable from that of M_8, using the following (straightforward) series of arguments:

1. M_0 and M_1 only differ in how c is constructed. It follows from the pseudorandomness of outputs of sbcom_τ that the output of M_0 is indistinguishable from that of M_1. Observe that if V^*'s leakage was more than n^2 bits, S receives H, and continues like M_1. In this situation, the lemma already follows. We therefore assume that the answer y returned $\mathcal{L}_H^n$ (if any), is such that $|y| \leq n^2 \leq |r| - n$. In addition, since the leakage queries F are polynomial-size circuits defined over c, the indistinguishability holds even in presence of such leakage. This argument about leakage queries will remain unchanged until hybrid M_7.

2. The fact that C_{pub} is a fake public-coin sender for C implies that the output of M_1 is computationally indistinguishable from that of M_2. This is because interactions with C_{pub} are indistinguishable from interactions with C (see section 3.2). Observe that string ω is indeed a valid witness for σ.

3. Indistinguishability of the outputs of M_3 and M_2 follows from the indistinguishability of the message $R_{\mathrm{OT}}(1^n, b)$ from $R_{\mathrm{OT}}^{pub}(1^n, p)$ (see section 3.1). If not, then we can first define n intermediate hybrids, fix one execution $i \in [n]$ of OT to receive the OT-message from outside, and contradict the hypothesis that $R_{\mathrm{OT}}(1^n, b) \overset{\mathrm{c}}{\equiv} R_{\mathrm{OT}}^{pub}(1^n, p)$, which is a contradiction.

4. The soundness of szkaok ensures that except with negligible probability, hybrid M_4 does not output $\mathtt{fail}_1$. Therefore, output distributions of M_3 and M_4 are statistically close.

5. From the pseudorandomness of prg it follows that the outputs of M_5 and M_4 are computationally indistinguishable. If not, then there exists a distinguisher D for prg. Briefly, define $M_4 : i$ for $i = 0, \ldots, n^3$ so that $M_{4:0} = M_4, M_{4:n^3} = M_5$, and $M_{4:i}$ changes only one component of t_2 as described in the simulator (instead of them all at the same time). Then, for any fixed i, D on input a challenge a internally runs $M_{4:i}$ and sets bits of t_2 so that $\mathsf{t}_i = \mathsf{prg}(s^1) \oplus a$ for a random s^1. It sends $\mathsf{prg}(s^1)$ to commit to 0, and a to commit to 1. It is easy to see that D's advantage is polynomially related to the distinguishing advantage between M_4 and M_5.

6. Outputs of M_5 and M_6 are indistinguishable since M_6 outputs a message containing $\mathtt{fail}_2$ with negligible probability. If not, then computational binding of shcom is broken: the output contains a different opening of either $\widehat{t_1}$ or $\widehat{ch}$, in addition to the one already known to the simulator.

7. Note that the output of M_7 contains the view v output by Blum's simulator, which is indistinguishable from the real view in M_6. In addition, since $\mathcal{R}$ outputs uniform randomness conditioned on the view being v, it holds that the leakage queries are also answered (and distributed) correctly. Therefore, outputs of M_6 and M_7 are also indistinguishable. Finally, observe that outputs of M_7 and M_8 are identically distributed.

This completes the proof. ■

References

[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. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

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

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pages 119–135, 2001.

[Ajt11] Miklós Ajtai. Secure computation with information leaking to an adversary. In *STOC*, pages 715–724, 2011.

[AK97] Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In *Security Protocols Workshop*, pages 125–136, 1997.

[Bar01] B. Barak. How to go beyond the black-box simulation barrier. In *Proc. 42nd FOCS* [foc01], pages 106–115.

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *FOCS*, 2002.

[BCH12] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In *TCC*, pages 266–284, 2012.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *Annual IEEE Conference on Computational Complexity (CCC)*, volume 17, 2002. Preliminary full version available as Cryptology ePrint Archive, Report 2001/105.

[BGGL01] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-sound zero-knowledge and its applications. pages 116–125, 2001.

[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.

[BL04] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. *SIAM Journal on Computing*, 33(4):783–818, August 2004. Extended abstract appeared in STOC 2002.

[Blu87] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987.

[BSW11] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In *EUROCRYPT*, pages 89–108, 2011.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Werner [foc01], pages 136–147. Preliminary full version available as Cryptology ePrint Archive Report 2000/067.

[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.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *STOC*, pages 542–552, 1991.

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, 2009.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DHLAW10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DHLAW10b] 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.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *CRYPTO*, pages 432–450, 2000.

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

[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.

[foc01] *Proc. 42nd FOCS*, 2001.

[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.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai. Leakage resilient zero knowledge. In *Advances in Cryptology – CRYPTO*, 2011.

[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.

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

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In ACM, editor, *Proc. 19th STOC*, pages 218–229, 1987. See [Gol04, Chap. 7] for more details.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991. Preliminary version in FOCS' 86.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications.* Cambridge University Press, 2004.

[HK12] Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. *J. Cryptology*, 25(1):158–193, 2012.

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

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short pcps. In *IEEE Conference on Computational Complexity*, pages 278–291, 2007.

[IKOS10] Yuval Ishai, Abishek Kumarasubramanian, Claudio Orlandi, and Amit Sahai. On invertible sampling and adaptive security. In *ASIACRYPT*, pages 466–482, 2010.

[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.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc. 24th STOC*, pages 723–732, 1992.

[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.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *Crypto '01*, pages 171–189, 2001.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LRW11] Allison B. Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.

[LW10] Allison B. Lewko and Brent Waters. On the insecurity of parallel repetition for leakage resilience. In *FOCS*, pages 521–530, 2010.

[Mic94] S. Micali. CS proofs. In *Proc. 35th FOCS*, pages 436–453, 1994.

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

[MR91] Silvio Micali and Phillip Rogaway. Secure computation. In *Crypto '91*, pages 392–404, 1991.

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

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[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.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proc. 36th STOC*, pages 232–241, 2004.

[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.

[Ros00] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *Crypto '00*, pages 451–468, 2000.

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

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167, 1986.

A Standard Definitions and Protocols

A.1 Security of Oblivious Transfer

Definition 4 (Oblivious Transfer [HK12]). *Let $\ell(\cdot)$ be a polynomial and $n \in \mathbb{N}$ the security parameter. A two-message, two-party protocol $\langle S_{\mathrm{OT}}, R_{\mathrm{OT}} \rangle$ is said to be a* statistically secure oblivious transfer *protocol for bit-strings of length $\ell(n)$ such that both the sender S_{OT} and the receiver R_{OT} are* PPT *interactive Turing machines receiving 1^n as common input; in addition, S_{OT} gets as input two strings $(m_0, m_1) \in \{0,1\}^{\ell(n)} \times \{0,1\}^{\ell(n)}$ and R_{OT} gets as input a choice bit $b \in \{0,1\}$. We require that the following conditions are satisfied:*

- Functionality: *If the sender and the receiver follow the protocol then for every $n \in \mathbb{N}$, every $(m_0, m_1) \in \{0,1\}^{\ell(n)} \times \{0,1\}^{\ell(n)}$, and every $b \in \{0,1\}$, the receiver outputs m_b.*

- Receiver security: *Denote by $\{R_{\mathrm{OT}}(1^n, b)\}_{n\in\mathbb{N}}$ denotes the message sent by honest receiver on input $(1^n, b)$. Then, the ensembles $\{R_{\mathrm{OT}}(1^n, 0)\}_{n\in\mathbb{N}}$ and $\{R_{\mathrm{OT}}(1^n, 1)\}_{n\in\mathbb{N}}$ are computationally indistinguishable; $\{R_{\mathrm{OT}}(1^n, 0)\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{R_{\mathrm{OT}}(1^n, 1)\}_{n\in\mathbb{N}}$*

- Sender security: *Denote by $S_{\mathrm{OT}}(1^n, m_0, m_1, q)$ the response of the honest sender with input $(1^n, m_0, m_1)$ when the receiver's first message is q. Then, there exists a negligible function* negl$(\cdot)$ *such that for every three messages $m_0, m_1, m' \in \{0,1\}^{\ell(n)} \times \{0,1\}^{\ell(n)}$, and every message $q \in \{0,1\}^*$ (from a possibly cheating, not necessarily polynomial time, receiver), and every sufficiently large $n \in \mathbb{N}$, it holds that either*

$$\Delta(S_{\mathrm{OT}}(1^n, m_0, m_1, q), S_{\mathrm{OT}}(1^n, m_0, m')) \leq \mathrm{negl}(n)$$

  or

$$\Delta(S_{\mathrm{OT}}(1^n, m_0, m_1, q), S_{\mathrm{OT}}(1^n, m', m_1)) \leq \mathrm{negl}(n)$$

  where $\Delta(X, Y)$ denotes the statistical distance between distributions X and Y.

A.2 Extractable Commitment

We start by recalling the protocol of [BL04], and then explain our (minor) changes. The protocol assumes the existence of trapdoor one-way permutations.

Protocol $\langle \mathsf{C}', \mathsf{R}' \rangle$. Security parameter is n. Committer's private input is a bit b; receiver has no private input. The steps of the protocol are as follows:

1. (a) C' samples a trapdoor one-way permutation (f, f^{-1}) and sends f to R'. C' also sends a random string τ for sbcom.

   (b) C' proves to R' using a standard zero-knowledge proof that f is a permutation.

2. R' commits to r_1 by sending $x_1 \leftarrow \mathsf{sbcom}_\tau(r_1; s)$.

3. C' sends a random string r_2.

4. R' sends r_1, and proves to C' that there exists s such that $x_1 = \mathsf{sbcom}(r_1; s)$ using a straight line szkaok protocol such as Π_{PR}. C' aborts if the proof fails.

5. C' sends $\widetilde{b} = b \oplus \mathsf{hcb}(f^{-1}(r_1 \oplus r_2))$, where hcb is the *hard core bit* function.

It is easy to see that the protocol is *perfectly*-binding and computationally-hiding. To open, the committer sends a string d such that $f(d) = r_1 \oplus r_2$. By repeating this protocol in parallel for every bit to be committed, and using only a single szkaok for them in step 3, we get a string a commitment scheme.

We now recall the extraction procedure given in [BL04], to satisfy the commit-with-extract property. The extraction procedure outputs a pair (υ, b), where υ is the (simulated) view of *a cheating committer (or sender)*. It holds w.h.p. that if υ is accepting then b is the bit committed in υ, otherwise $b = \perp$. The extraction procedure E' starts by interacting with a cheating committer, say C'', and sends messages according to algorithm R' up to step 3. In step 4, it selects a string d and sets $r_1 = f(d) \oplus r_2$; to successfully complete this step, it uses the simulator S_{PR} of szkaok. C'' either responds with a bit $\tilde{b}$ or aborts, to complete the (simulated) view υ; E' outputs (υ, b) where $b = \mathsf{hcb}(d) \oplus \widetilde{b}$ if C'' does not abort, and $\perp$ otherwise. The correctness of this extractor is easy to verify, see [BL04] for details.

Modifications. We now describe the required changes to make this protocol suitable for our usage. We need to make the following changes:

- The use of step 1(b) is problematic since it will not allow a fake public-coin receiver algorithm. Therefore we would like to use a trapdoor permutation which can be easily verified to be a permutation. In addition, description of f should just be a truly random string/element. Existence of such trapdoor permutations suffices. Unfortunately we do not know any candidates.[15]

- We therefore replace the entire step 1 as follows: C' sends the public-key of ElGamal system, and keeps the secret key as the trapdoor. While this will not be a permutation, it will still satisfy our goals. Note that ElGamal has verifiable public keys. We work with safe primes $p = 2q + 1$ and in an order q subgroup G_q of $\mathbb{Z}_p^*$.

- In step 3, the string $r_1 \oplus r_2$ is used to sample two random elements in G_q which define the encryption of a random element, say r. The secret-key is used to recover this element and then commit to b using r (e.g., using hcb function). These changes are already sufficient to ensure that there is a fake public-coin committer for this protocol.

- The only remaining property is statistical simulation. Since r_1 is committed using a perfectly binding commitment scheme, it ends up in only a computationally indistinguishable view υ for the sender. To get statistical simulation, we replace this sbcom, by shcom. The committer will now send ρ (for shcom) instead of τ (for sbcom) in the first step to implement this change.

It is easy to verify that these changes do not violate either the perfect binding, or computational-hiding of the protocol. In addition, the simulation and extraction of the committed value proceeds as before. We denote this resulting scheme by $\Pi_{com}; = \{\mathsf{C}, \mathsf{R}\}$. The scheme also has public decommitment, which consists of the C simply sending the (unique) secret-key of the ElGamal public-key.

Finally, note that the fake public-sender C_{pub} consists of simply sending either random strings or the public-key which is two random elements in G_q.

B Missing Proofs

Proof of lemma 1. Let $\Pr[\sigma]$ denote the probability that the prefix σ occurs in interaction of honest V with P^*, and $\Pr[P^*_\sigma]$ denote the success probability of P^*_σ. Let $\mathcal{G}$ be the set of σ such that

[15]We cannot use factor-based constructions, since $N = pq$ is distinguishable from things that can be sampled using public coins; likewise no *trapdoor* permutation based on discrete log either are known.

$\Pr[P^*_\sigma]$ is at least $\epsilon_n/2$. Then,

$$\begin{aligned}
\epsilon_n &= \sum_{\sigma} \Pr[\sigma] \cdot \Pr\left[P^*_\sigma\right] && (3)\\
&= \sum_{\sigma \in \mathbf{L}_{sim}} \Pr[\sigma] \cdot \Pr\left[P^*_\sigma\right] + \sum_{\sigma \in \mathcal{G} \wedge \sigma \notin \mathbf{L}_{sim}} \Pr[\sigma] \cdot \Pr\left[P^*_\sigma\right] + \sum_{\sigma \notin \mathcal{G} \wedge \sigma \notin \mathbf{L}_{sim}} \Pr[\sigma] \cdot \Pr\left[P^*_\sigma\right] && (4)\\
&\leq \operatorname{negl}(n) \cdot 1 + \Pr\left[\sigma \in \mathcal{G} \wedge \sigma \notin \mathbf{L}_{sim}\right] \cdot 1 + \frac{\epsilon_n}{2} \cdot 1 && (5)
\end{aligned}$$

Therefore $\Pr\left[\sigma \in \mathcal{G} \wedge \sigma \notin \mathbf{L}_{sim}\right] \geq \epsilon_n/2 - \operatorname{negl}(n) \geq \epsilon_n/4$, as desired. ■