Non-black-box Techniques Are Not Necessary for Constant Round Non-malleable Protocols

Omkant Pandey
omkant@cs.ucla.edu

University of California, Los Angeles

Abstract

Recently, non-black-box techniques have enjoyed great success in cryptography. In particular, they have led to the construction of *constant round* protocols for two basic cryptographic tasks (in the plain model): non-malleable zero-knowledge (NMZK) arguments for NP, and non-malleable commitments. Earlier protocols, whose security proofs relied only on black-box techniques, required non-constant (e.g., $O(\log n)$) number of rounds. Given the inefficiency (and complexity) of existing non-black-box techniques, it is natural to ask whether they are *necessary* for achieving constant-round non-malleable cryptographic protocols.

In this paper, we answer this question in the *negative*. Assuming the validity of a recently introduced assumption, namely the *Gap Discrete Logarithm* (GAP-DL) assumption [MMY06], we construct a constant round *simulation-extractable* argument system for NP, which implies NMZK. The GAP-DL assumption also leads to a very simple and natural construction of *non-interactive non-malleable commitments*. In addition, plugging our simulation-extractable argument in the construction of Katz, Ostrovsky, and Smith [KOS03] yields the first $O(1)$-round secure multiparty computation with a dishonest majority using only black-box techniques.

Although the GAP-DL assumption is relatively new and non-standard, in addition to answering some long standing open questions, it brings a new approach to non-malleability which is simpler and very natural. We also demonstrate that GAP-DL holds unconditionally against *generic* adversaries.

1 Introduction

Most security proofs in cryptographic literature can be seen as having one of the following two flavors: *black-box* and *non-black-box*. Black-box proofs of security are those which use/access the adversary A (if at all) *only as an oracle*. Non-black-box proofs of security, however, use the adversary in more ways than just as an oracle. For example, such security proofs may use the actual *code* of the adversary. Such security proofs are also referred as non-black-box methods/techniques. Non-black-box techniques can sometimes achieve goals that cannot be achieved using black-box techniques [Bar01, BGGL01, BL04, MP06]. An important research direction in theoretical cryptography is to understand where a non-black-box proof of security is *necessary*. The importance of this question stems from the fact that black-box techniques are typically the "natural" way of thinking about security proofs. In addition, they also seem to give rise to more efficient protocols.

Non-black-box techniques have recently enjoyed great success, especially in the area of *zero knowledge* [GMR85, GMW87]. Barak [Bar01] demonstrated how to use the code of a cheating verifier V^* to construct a public coin, negligible soundness error, constant round zero knowledge argument system for all languages in NP. Further, Barak's protocol admits *strict* polynomial time simulation, and remains zero-knowledge even when composed *concurrently* for a *bounded* polynomial number of executions [FS90, DNS98]. It is known that a protocol with such properties cannot be proven secure using black-box techniques [Gol01, CKPR03, Bar01, BL04]. The use of non-black-box techniques is thus essential here.

Non-malleable Protocols. Non-black-box techniques, later, proved to be immensely useful in reducing the round complexity of the so called *non-malleable* cryptographic protocols [DDN00]. Consider a man-in-the-middle, A, who interacts with the prover P of a zero-knowledge protocol π and receives the proof of a theorem, x. A simultaneously participates in another interaction of π with some other entity who acts as a verifier V, to prove the validity of some, possibly *related*, theorem $\tilde{x}$. Informally, the notion of *non-malleable zero-knowledge* (NMZK) says that for convincing V, A does not benefit from interacting with P. That is, the probability of A successfully convincing V of the validity of $\tilde{x}$ when A interacts with P, is essentially the same when it interacts with a *simulator* S who does not have a witness for x (see section 2 for formal definitions). Such proof systems were constructed by [DDN00] using only *black-box* techniques, albeit in a non-constant ($O(\log n)$) number of rounds.[1]

Not much progress was made on reducing the round complexity of NMZK until Barak [Bar02], who constructed a *constant* round NMZK protocol for all of NP using non-black techniques and assuming super-polynomial hardness assumptions. Pass [Pas04], and Pass and Rosen [PR05] improved upon this by achieving the same result under (standard) polynomial hardness assumptions. All of these works, however, still rely on non-black-box techniques.

Unlike (bounded) concurrent zero-knowledge, where non-black-box techniques are *necessary* to reduce the round complexity below $\Omega\left(\frac{\log n}{\log \log n}\right)$ [CKPR03] (see also [MY08]), no such impossibility result exists for NMZK. It is thus natural to ask, whether non-black-box techniques are *necessary* for constructing constant round NMZK protocols for NP? That is,

Do constant round NMZK *protocols (for* NP*), with a* black-box *proof of security exist?*

The notion of non-malleability prevails in other cryptographic primitives too, such as commitment schemes. Briefly and very informally, a commitment scheme is said to be non-malleable if a man-in-the-middle A receiving a commitment to a value v, cannot commit to a *related* value $\tilde{v}$. That is, $\tilde{v}$ is *computationally independent* from v. An $O(\log n)$-round commitment scheme with a black-box proof of security [DDN00], and an $O(1)$-round scheme with a non-black-box proof of security [PR05][2] are known to exist for this task. The following intriguing question has, however, remained (completely) unanswered,

Do non-interactive *and non-malleable (*NINM*) commitments exist in the* plain *model?*

These two questions (of constructing constant-round black-box NMZK arguments, and NINM commitments) are among a few *long standing* open problems in cryptography. In this paper, we answer both of these questions in the *affirmative*.

- We construct a constant round *simulation-extractable* argument system for NP, which admits a black-box proof of security. *Simulation-extractability* [Sah99, DDO+01] (see also [PR05]), is a stronger (and more useful) property which implies NMZK. (For a formal statement of this result, see theorem 7 in section 4).

- We construct (bit and string) commitment schemes that are non-interactive and non-malleable with respect to commitment (NINM). These constructions are simple and natural. (For formal statements, see theorem 12 and theorem 13 in section 5).

Katz, Ostrovsky, and Smith [KOS03] constructed an $O(1)$-round protocol for securely computing any functionality even when $n-1$ out of n participants in the protocol are malicious (assuming standard point-to-point perfect (authenticated, secret, unjammable) and synchronous channels with broadcast). They use

[1] In this paper, we always work in the plain model which makes no set-up assumptions.

[2] In [PR05] a simulation based definition of non-malleability is used; the simulation in their case is black-box, but the *analysis* relies on non-black-box techniques.

a slightly modified version of Barak's protocol [Bar02], which results in a non-black-box proof of security. By plugging-in our simulation-extractable protocol in their construction, we are able to achieve the same result, using only *black-box* techniques.

Our results are based on a relatively non-standard assumption, namely the *Gap Discrete Logarithm* (GAP-DL) assumption. This assumption was first introduced by Malkin, Moriarty, and Yakovenko [MMY06].[3] However, assumptions of similar sort have previously existed. They were first conceived by Okamoto and Pointcheval [OP01a, OP01b]; and have regularly been used thereafter, for obtaining security proofs of various cryptographic (e.g., signatures, encryption, authentication, etc.) schemes [JJ02, BSZ07, KP05, SWP04, ABR01]. In fact, sometimes they are essential, see [SBZ02]. A similiar assumption about *collision resistant hash functions* was recently used by Prabhakarn and Sahai [PS04] to obtain UC-secure [Can01] multiparty computation under "super polynomial" time simulation [Pas03].

Informally, the assumption states that given a sufficiently large safe prime p, taking discrete logarithms is hard in (prime order) subgroup of $\mathbb{Z}_p^*$ for every efficient adversary A^* *even when* it is given access to an oracle, $\mathcal{O}_p$, which computes discrete logarithms in prime order subgroup $\mathbb{Z}_q^*$ such that q is a safe prime *different* from p, i.e., $q \neq p$. Although this assumption is relatively new and less well studied, assuming its validity we are able to resolve two long standing open questions in cryptography.

We gain more confidence in the GAP-DL assumption by measuring its strength against *generic* algorithms. We prove that the GAP-DL assumption holds *unconditionally* in the "generic" group model of Shoup [Sho97]. A generic algorithm is one that does not make use of the specific *encoding* of group elements. Group elements appear to the algorithm to be encoded as arbitrary *unique* strings, so that no property other than equality can be tested directly (see section 7 and theorem 15). This result, however, has no bearing with respect to non-generic algorithms, such as the index-calculus method.

Finally we would like to remark that the level of complexity in current constructions of secure non-malleable protocols is a (growing) concern. The security proofs in this area typically turn out to be quite complex and subtle. The GAP-DL assumption offers a new approach to non-malleability, which is simpler and more natural.

Techniques. In our work, we look at the construction of the constant-round simulation-extractable argument as a *building block*, rather than an end goal. Further, we focus on *simulation-extractability* and not just on non-malleability. This leads to a construction that is independent from the construction of NINM commitments. The construction uses a new type of *2-slot* simulation and *2-slot* extraction. In addition, it seems to us that due to the presence of an oracle, the proof of simulation-extractability of our protocol is slightly different from the usual proofs. These techniques might be of independent interest.

Remark. We remark that our work considers *standard* (i.e., expected polynomial time) simulation and extraction, in the *plain* model. The oracle for solving discrete logarithms *is never* available during simulation/extraction. A confusion should be avoided with"quasi-polynomial" time simulation/extraction, where the simulator/extractor is allowed to run in time that is slightly more than a polynomial (quasi-polynomial) [Pas03]. In earlier works (e.g., [PS04]) the oracle *enters* simulation/extraction, which leads to super-polynomial running time. A main novelty of this work is in acheiving standard simulation/extraction.

Related Work. In addition to the works discussed above, several works have previously considered constructing NMZK arguments and NINM commitments in settings where a trusted setup is available. Perhaps most popular among these is the CRS-model [BSMP91] where a common-reference-string (CRS) is generated by a trusted third party according to a prespecified distribution during the system setup. The CRS is then

[3] Our actual assumption is, in fact, *weaker* than the assumption of [MMY06], see section 3. Also, [MMY06], call this assumption as *relativizing* DLA (rDLA). We prefer to call it GAP-DL to remain consistent with previously existing literature [OP01a].

made available to all the participants. In this model, NMZK *proofs* can be achieved in one-round (i.e., non-interactively) [Sah99] (see also [DDO+01]). In the context of commitment schemes, NINM commitments in the CRS-model were first constructed by [DIO98] (see also [FF00, DKOS01]).

2 Definitions

We present definitions relevant to our work. We assume that the reader is familiar with the following: *perfectly hiding* and *perfectly binding* notions of commitment schemes, interactive proofs/arguments, and statistical/perfeft witness indistinguishability (see [Gol01]). In the following, κ denotes the security parameter, and L is an NP-language with witness relation R_L. A statement x is in L iff there exists a w of length $\text{poly}(|x|)$ such that $R_L(x, w) = 1$.

Non-malleable Zero-Knowledge. The man-in-the-middle setting proceeds as follows. First, the input of the honest prover P, i.e., statement $x \in L \cap \{0,1\}^n$ is chosen. P is then given a witness w (s.t.$R_L(x, w) = 1$), with a uniformly chosen random tape. The man-in-the-middle, A may now start interacting with P while playing the role of a verifier of the protocol π (in consideration). This is called the "left" interaction. At any point, A, may adaptively output a new statement $\tilde{x} \in L \cap \{0,1\}^n$. At this point, an honest verifier V, is created with input $\tilde{x}$ and uniformly chosen randomness. A can now interact with V simultaneously, attempting to prove the validity of $\tilde{x}$ by playing the role of the prover. Interaction of A with V is called the "right" interactions. The adversary A is a non-uniform PPT machine with some auxiliary information.

Following [PR05], we work with a somewhat stronger notion called *simulation-extractability*, which implies non-malleable zero-knowledge argument (proof) of knowledge property. The formulations that we present here are taken from [BPS06].

Definition 1 *A protocol* $\pi \stackrel{def}{=} \langle P, V \rangle$ *is said to be* simulation-extractable *for membership in an* NP *language* L *with witness relation* R_L*, if it is an interactive argument system between a prover and verifier (both* PPT*) such that the following conditions hold.*

Completeness *For every* x, w *such that* $R_L(x, w) = 1$*,* $P(x, w)$ *makes* V *accept with probability 1.*

Simulation-Extractability *For every (non-uniform)* PPT*adversary* A *(with auxiliary information* aux*) launching a man-in-the-middle attack as above (i.e.,* A *interacts with a* P *on "left" and a* V *on right as defined above), there exists an expected polynomial time simulator-extractor* $\text{SE} = (S, K)$ *outputting* $(\nu, \tilde{w})$ *on input* (x, aux) *such that,*

- ν *is the simulated (joint) view of* A *and* V*, such that* ν *is computationally indistinguishable from the view of* A *in a real execution.*
- *Let* $\mathsf{trans}, \mathsf{trans}'$ *denote the transcripts of left and right executions respectively. If the right interaction is accepting, with statement* $\tilde{x}$*, and* $\mathsf{trans} \neq \mathsf{trans}'$*,* $R_L(\tilde{x}, \tilde{w}) = 1$ *except with negligible probability.*

The probability is taken over the random coins of SE*. Further, the protocol is* black-box *simulation-extractable, if* SE *is a universal machine that uses* A *only as an oracle, i.e.,* $\text{SE} = \text{SE}^M$*.*

The condition $\mathsf{trans} \neq \mathsf{trans}'$ in the definition says that if the right session is not an *exact copy* of the left session, then SE should output a valid witness for the statement of the right session, whenever this session is accepting.

Non-malleable Commitments. In case of commitments, the man-in-the-middle scenario is exactly the same except that the prover P is replaced by a committer C, and the verifier V is replaced by a receiver R. There are no theorems $x, \tilde{x}$. Instead, A receives a commitment to either $v \in \{0,1\}^l$ (of its choice) or 0^l from C, and commits to a value $\tilde{v}$ to R such that $\tilde{v}, v$ are *related* (according to some polynomial time computable relation). As we are dealing with *non-interactive* commitments, A receives a single message from C.

Definition 2 *A perfectly binding non-interactive commitment scheme $\langle C, R\rangle$ is said to be* non-malleable with respect to commitment *if for every (non-uniform)* PPT *adversary A with auxiliary information aux, the outputs of the following two experiments are computationally indistinguishable.*

Experiment 1. *A first outputs a string $v \in \{0,1\}^l$. A commitment c to the value v is generated honestly according to the algorithm C. A receives c, and outputs a string $\tilde{c}$. If $\tilde{c} = c$ or if algorithm R rejects $\tilde{c}$, the output of the experiment is $\perp$. Otherwise, (i.e., R accepts $\tilde{c}$), $\tilde{c}$ defines a unique commitment string $\tilde{v}$; the output of the experiment is $\tilde{v}$.*

Experiment 2. *A first outputs a string $v \in \{0,1\}^l$. A commitment c to the value 0^l is generated honestly according to the algorithm C. A receives c, and outputs a string $\tilde{c}$. If $\tilde{c} = c$ or if algorithm R rejects $\tilde{c}$, the output of the experiment is $\perp$. Otherwise, (i.e., R accepts $\tilde{c}$), $\tilde{c}$ defines a unique commitment string $\tilde{v}$; the output of the experiment is $\tilde{v}$.*

This definition is slightly more convenient to work with, and implies simulation style definitions [PR05].

Hard-core Bits. We assume that the reader is familiar with the notion of *hard-core bits*. The hard-core bit of a *one-way function* f, corresponding to an input x from its domain will be denoted by $\text{HCB}_f(x)$.

Blum's Hamiltonicity Protocol. Blum's Hamiltonicity (BH) protocol, is a 3-round honest verifier zero-knowledge protocol for proving that a graph $G = (V, E)$ contains a hamiltonian cycle C. Further, the protocol is *special-sound*, which means that given two accepting transcripts (a, b, c) and (a', b', c') such that $a = a', b \neq b'$, the cycle C can be computed in polynomial time. We will use the parallel version of the protocol, which is a *witness indistinguishable argument of knowledge* with negligible soundness error. If the protocol uses a perfectly hiding commitment scheme, the protocol is witness independent.

Strong Signatures. A signature scheme $(\mathcal{K}, \text{SIGN}, \text{VERIFY})$ is said to be *strongly unforgeable* if no efficient adversary, with access to a signing oracle with respect to verification key VK, can output a pair (m, σ) with non-negligible probability, such that: $\text{VERIFY}(m, \sigma, \text{VK}) = 1$ and the pair (m, σ) does not correspond to the input-output pair of a performed oracle query. A *strong signature scheme* is a signature scheme that is strongly unforgeable.

Invertible Reductions. Let $H = \{G : G \text{ is hamiltonian}\}$ be the NP-complete language of graph hamiltonicity, i.e., it contains graphs G such that G has a hamiltonian cycle. Let $R_H \subseteq \{0,1\}^* \times \{0,1\}^*$ be the following relation for H: $(G, C) \in R_H$ if and only if C is a hamiltonian cycle in the graph G. We write $R_H(G, C) = 1$ iff $(G, C) \in R_H$, and $C \in R_H(G)$ iff $R_H(G, C) = 1$. Relation R_H is an NP-complete relation which comes with two polynomial time computable and non-shrinking functions $f : \{0,1\}^* \to \{0,1\}^*$ and $g : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that for every NP-relation R' it holds that: $x \notin L_{R'} \Rightarrow f(x) \notin H$ and $(x, w) \in R' \Rightarrow (f(x), g(x, w)) \in R_H$. Here $L_{R'} = \{x : \exists w \text{s.t.} (x, w) \in R'\}$. Further, f, g are *polynomial time invertible* functions with inverses f^{-1}, g^{-1} respectively, and we say that R_H is NP-complete *via invertible reductions.*

Notation. Throughout the paper, $\text{neg} : \mathbb{N} \to \mathbb{R}$ denotes a negligible function in κ (the security parameter). If a message u appears in the "left" session, then its counterpart in the "right" session will be denoted by $\tilde{u}$. We will frequently use a perfectly hiding commitment scheme denoted by $(\text{COM}_{\text{PH}}, \text{DCOM}_{\text{PH}})$. Such schemes can be constructed from a variety of assumptions, such as the Decisional-Diffie-Hellman assumption, in only *two* rounds (see, for example [Ped91]).

3 The Gap Discrete Logarithm Assumption

For convenience, we work with safe-primes. Let p, q be two primes such that $p = 2q + 1$. The order q subgroup of the safe prime p will be denoted by G_q. Let $|p| = \kappa$, where κ is a (sufficiently large) security parameter, and let g be random generator of G_q.

The Discrete Logarithm Assumption (DLA). Let $y = g^a \mod p$, for a uniformly chosen $1 \leq a \leq q-1$. We say that DLA holds in G_q if for all non-uniform expected polynomial time adversaries $\mathcal{A}$ (with random coins ω) and sufficiently large κ, it holds that

$$\Pr_{p,g,a,\omega}[\mathcal{A}(p, g, y; \omega) = a] \leq \text{neg}(\kappa)$$

For this paper, triplet (p, g, y) is called a DL-instance, iff $p = 2q + 1$ such that q is a prime, g generates G_q, and $y \in G_q$. The GAP-DL assumption is the same as DLA except that adversary $\mathcal{A}$ is also given access to a oracle, $\mathcal{O}_{p,\ell(\kappa)}$. Oracle $\mathcal{O}_{p,\ell(\kappa)}$ solves DL-instances (p', g', y') for the adversary $\mathcal{A}$, as long as $p' \neq p$. We actually work with a slightly weaker version by enforcing a stronger requirement than $p' \neq p$, namely that p' should be at least $\ell(\kappa)$ "far" from p. By "far" we mean that the sizes of p and p' differ at least by $\ell(\kappa)$. The formal description of the oracle follows.

Oracle $\mathcal{O}_{p,\ell(\kappa)}$. For security parameter $\kappa \in \mathbb{N}$, function $\ell : \mathbb{N} \to \mathbb{N}$, and a safe prime $p = 2q + 1$ such that $\ell(\kappa) \leq |p| \leq (2\kappa + 1) \cdot \ell(\kappa)$, define the oracle $\mathcal{O}_{p,\ell(\kappa)}$ which acts as follows. On input a query $Q_i = (p'_i, g_i, y_i)$, if the following two conditions hold,

1. The query Q_i is a DL-instance. That is, $p'_i = 2q'_i + 1$ is a safe prime, g_i generates $G_{q'_i}$, and $y_i \in G_{q'_i}$.
2. $\ell(\kappa) \leq |p'_i| \leq |p| - \ell(\kappa)$ or $|p| + \ell(\kappa) \leq |p'_i| \leq (2\kappa + 1) \cdot \ell(\kappa)$.

then, output a_i such that $g_i{}^{a_i} = y_i \mod p'_i$. Output $\perp$ otherwise.

The Gap Discrete Logarithm Assumption (Gap-DL). We say that GAP-DL holds G_q with respect to function $\ell(\cdot)$, if for all non-uniform expected polynomial time adversaries $\mathcal{A}$ (with random coins ω) and sufficiently large κ, it holds that

$$\Pr_{p,g,a,\omega}\left[\mathcal{A}^{\mathcal{O}_{p,\ell(\kappa)}}(p, g, y; \omega) = a\right] \leq \text{neg}(\kappa)$$

Notation. Let L_{DL} be the language containing DL-instances. Note that L_{DL} is actually easy to decide. We will be interested in the *discrete-log* relation R_{DL} for this language, which is defined as follows. For a DL-instance $\text{DL}_i = (p_i, g_i, y_i)$ and integer a, $R_{\text{DL}}(\text{DL}_i, a) = 1$ iff $y_i = g_i^a \mod p_i$ and $1 \leq a \leq \frac{p_i - 3}{2}$. Also, define the language $L_{\overline{\text{DL}}}$ which contains κ-dimensional vectors of DL-instances, $\overline{\text{DL}} = (\text{DL}_1, \ldots, \text{DL}_\kappa)$. Define the corresponding relation $R_{\overline{\text{DL}}}$ for this language analogously: $R_{\overline{\text{DL}}}(\overline{\text{DL}}, \overline{a}) = 1$ iff for all $1 \leq i \leq \kappa$ it holds that $R_{\text{DL}}(\text{DL}_i, a_i) = 1$ and $\overline{\text{DL}}, \overline{a}$ are κ-dimensional.

4 A Simulation-Extractable Argument

In our protocol, the prover P uses a verification-key VK of a strong signature scheme in a crucial manner, to achieve simulation-extractability. In more detail, it generates various prime numbers based on the string VK, according to the algorithm, PRIMES described next (assume w.l.o.g. $|\text{VK}| = \kappa$).

Algorithm PRIMES(VK): For $i = 1, \ldots, |\text{VK}|$, do the following: define $t_i = i \circ \text{VK}_i$ (thus, $|t_i| = 1 + \lceil \log \kappa \rceil$); randomly select a *safe* prime $p_i = 2q_i + 1$ such that $|p_i| = (t_i + 1) \cdot \ell(\kappa)$. Output the list $\vec{p} = (p_1, \ldots, p_\kappa)$.

Claim 3 *Let* $(p_1, \ldots, p_\kappa) \leftarrow \text{PRIMES}(\text{VK})$*, and* $(\tilde{p}_1, \ldots, \tilde{p}_\kappa) \leftarrow \text{PRIMES}(\widetilde{\text{VK}})$*. If* $\text{VK} \neq \widetilde{\text{VK}}$*, there exists* $i \in [\kappa]$ *such that for all* $j \in [\kappa]$*, it holds that* $\big||\tilde{p}_i| - |p_j|\big| \geq \ell(\kappa)$*.*

Proof. We have that $\tilde{t}_i = i \circ \widetilde{\text{VK}}_i$ and $t_j = j \circ \text{VK}_j$ for all $i, j \in [\kappa]$. By this construction, it follows that $\exists i : \forall j \;\; \tilde{t}_i \neq t_j$. We call this index i, the *target index*, and the associate prime $\tilde{p}_i$ the *target prime*. Further, for all j it holds that $|\tilde{t}_i - t_j| \geq 1$. Hence, we conclude that $\big||\tilde{p}_i| - |p_j|\big| \geq \ell(\kappa)$. ∎

Overview of Our Protocol. From strong-unforgeability of the signature scheme, it can be derived that if the man-in-the-middle A copies VK, it copies the entire left execution to its right execution (and hence essentially aborts). From claim 3, it is immediate that if A does not copy VK, i.e., the key $\widetilde{\text{VK}}$ on right is different from VK, then there exists a target prime. Consider what happens if we require the verifier of the protocol to set up DL-instances in all primes decided according to PRIMES(VK) so that V "knows" a solution to all these DL-instances. Because VK cannot be copied exactly, there would exist a target prime, for which A will be forced to "know" a solution, *irrespective of the "help" it may get by mauling* V's messages. That A, V "know" solutions to all DL-instances they generate, can be achieved by using an argument of knowledge. Later on, in the protocol, the prover P will execute an FLS-style argument of knowledge that "either it knows a witness to x or it knows a solution to all the DL-instances" [FLS99].

Above sketch runs into a slight problem if we try to prove its simulation-extractability. When we try to extract a witness for theorem $\tilde{x}$, we would like to argue that if extraction outputs a false witness (i.e., outputs a solution to DL-instances), then we can contradict GAP-DL assumption. For this, it would be crucial that we do not query the oracle on the challenge (or target) prime, while proving the security. But because extraction is black-box, it will have to rewind A. As a result, the left key VK may change in various threads, resulting in a change in the target prime, which may force illegal queries to the oracle, resulting in a failure in extraction.

To overcome this problem, we rewind A in a specific manner. First, we create *two* slots by requiring the prover to execute the FLS-style protocol twice. Depending on the scheduling of the messages, we now rewind A in one of these slots, such that the left key VK will not change during various rewindings. For the same reason, a similar approach is required when V proves the knowledge of solutions to all DL-instances: we use a Zero-Knowledge argument of knowledge which has a *2-slot* simulator, just like above. For some more technical reasons, we also need *statistical* security from these protocols.

The Actual Construction. A formal description of our constant round simulation-extractable argument system $\pi_{\text{SE}} : \langle P, V \rangle$ for NP appears in figure 1. Both parties P and V are PPT Turing Machines. The common input is a statement $x \in L_n$, for language $L \in \text{NP}$ with witness relation R_L, and $L_n = L \cap \{0,1\}^n$. P proves to V the knowledge of w such that $R_L(x, w) = 1$. The length parameter function of the protocol is denoted by $\ell : \mathbb{N} \to \mathbb{N}$. As discussed above, the protocol uses two *2*-slot protocols: π_{SZA} and π_{SWA}.

PROTOCOL π_{SZA}. This is a statistical zero-knowledge argument of knowledge (SZA) for NP, see figure 2. The common input to P and V is a statement $x \in L$, for some language L associated with an NP-relation R_L. P proves to V the knowledge of w such that $R_L(x, w) = 1$. The protocol uses NP-reductions to graph-hamiltonicity by using *invertible functions* (f, g). The special property of π_{SZA} is that its simulator has a

choice of slots in simulation. The *2*-slot simulator (S_{SZA}) and (normal) extractor (K_{SZA}) of this protocol is deferred until needed.

PROTOCOL π_{SWA}. This FLS-style protocol is a statistical witness-indistinguishable argument of knowledge (SWA) for NP, with *2*-slots for extraction (see figure 3). The common input to P and V is a graph $G = (V, E)$. P proves to V the knowledge of a Hamiltonian cycle C in G. The corresponding relation is denoted by R_H and $R_H(G, C) = 1$ if and only if C is a hamiltonian cycle in G. The protocol is just a *sequential* repetition of the classical BH-protocol (with negligible soundness error) *twice*.

In all steps above, whenever a message is not according to the protocol specifications, an honest party aborts the protocol.

$P \to V$: P Let $(\text{VK}, \text{SK}) \leftarrow \mathcal{K}(1^\kappa)$, Send VK to V. Assume, w.l.o.g, that $|\text{VK}| = \kappa$.

$V \to P$: Let $(p_1, \ldots, p_\kappa) \leftarrow \text{PRIMES}(\text{VK})$ so that $p_i = 2q_i + 1$. For $i = 1, \ldots, |\text{VK}|$, randomly select DL-instances $\text{DL}_i = (p_i, g_i, y_i)$, in subgroup G_{q_i} of $\mathbb{Z}^*_{p_i}$. Let $a_i \in \mathbb{Z}_{q_i}$ be s.t. $y_i = {g_i}^{a_i} \mod p_i$, $\overline{\text{DL}} = (\text{DL}_1, \ldots, \text{DL}_\kappa)$, $\overline{y} = (y_1, \ldots, y_\kappa)$, and $\overline{a} = (a_1, \ldots, a_\kappa)$. V sends $\overline{\text{DL}}$ to P.

$V \Leftrightarrow P$: V and P execute the protocol π_{SZA} on common input $\overline{\text{DL}}$, in which V proves to P the knowledge of $\overline{a}$ such that $R_{\overline{\text{DL}}}(\overline{\text{DL}}, \overline{a}) = 1$.

$P \Leftrightarrow V$: P proves to V the knowledge of w for statement z such that $\mathbf{R}(z, w) = 1$. Here, $z = (x, \overline{\text{DL}})$ and for $w = (w_1, w_2)$ we define the witness relation $\mathbf{R}$ as following,

$$\mathbf{R}(z, w) = (R_L(x, w_1)) \vee (R_{\overline{\text{DL}}}(\overline{\text{DL}}, w_2))$$

This is done by P and V executing the protocol π_{SWA} on common input $f(z)$. P (the prover of π_{SWA}) uses the witness $g(z, w)$ as its private input. (V sends the first message of a *perfectly hiding* commitment scheme before π_{SWA} begins, which is used by P.)

$P \to V$: Let trans be the transcript of communication so far. P sends $\sigma \leftarrow \text{SIGN}(\text{trans}, \text{SK}, \text{VK})$ to V.

Figure 1: Our $O(1)$-round Simulation-Extractable Argument System $\pi_{\text{SE}} : \langle P, V \rangle$

4.1 Proving Simulation-Extractability

We demonstrate that the protocol π_{SE} is a simulation-extractable argument. This is done by providing a simulator-extractor. Our simulator-extractor, SE, is a *black-box* machine running in (expected) *polynomial time*. Further, SE works in the *plain* model (in particular, without access to any oracle). For convenience, in the rest of the paper, let κ and n (input length) be polynomially related.

Let A' be a man-in-the-middle adversary who interacts with an honest prover, P, on the left while simultaneously interacting with an honest verifier, V, on right. Using a standard argument, it can be shown that if A' copies the verification key (also called the *tag* or the *identity*) from left to right, then with high probability (unless it aborts on right), it will also copy the entire left hand side conversation to the right hand side. Thus, we consider a modified man-in-the-middle adversary, A, which acts identically to A' except that whenever A' sets $\widetilde{\text{VK}} = \text{VK}$, our new adversary A aborts the right hand side execution. It is easy to see that our claims regarding the machine SE with respect to the adversary A, also hold with respect to the adversary A'.

Machine $\text{SE} = (S, K)$, consists of two machines – the simulator S and the extractor K. Simulator S will be used to simulate the joint view of A and V. If $\tilde{x}$ is the theorem in the right hand side interaction, then extractor K will be used to extract a witness for $\tilde{x}$ whenever V's view is accepting. We assume, without loss of generality, that A is deterministic. We now define some random variables.

$V \to P$: For $b \in \{0,1\}, j \in \{1,2\}$, and $i = 1, \ldots, \kappa$ V selects 4κ strings, each of length κ, uniformly at random: $\sigma^b_{j,i} \leftarrow \{0,1\}^\kappa$. Using a non-interactive *perfectly binding* commitment scheme V computes $c^b_{j,i} = \text{COM}_{\text{PB}}(\sigma^b_{j,i}; \omega^b_{j,i})$, where $\omega^b_{j,i}$ is the randomness (for the commitment). Define $\sigma_{j,i} = \sigma^0_{j,i} + \sigma^1_{j,i}$, and $c_{j,i} = (c^0_{j,i}, c^1_{j,i})$. V sends to P the vectors $\overline{c}_1 = (c_{1,1}, \ldots, c_{1,\kappa})$ and $\overline{c}_2 = (c_{2,1}, \ldots, c_{2,\kappa})$.

(Slot 1)

$P \to V$: Send $r_1 \leftarrow \{0,1\}^\kappa$ to V.

$V \to P$: For all $i = 1, \ldots, \kappa$, if $r_1[i] = b$ then V decommit to $c^b_{1,i}$ by sending $(\sigma^b_{1,i}, \omega^b_{1,i})$.

(Slot 2)

$P \to V$: Send $r_2 \leftarrow \{0,1\}^\kappa$ to V.

$V \to P$: For all $i = 1, \ldots, \kappa$, if $r_2[i] = b$ then V decommit to $c^b_{2,i}$, by sending $(\sigma^b_{2,i}, \omega^b_{2,i})$.

(Body)

$V \to P$: Send the first message of a 2-round *perfectly hiding* commitment scheme. This defines the instances of functions $(\text{COM}_{\text{PH}}, \text{DCOM}_{\text{PH}})$, to be used in the next round.

$P \Leftrightarrow V$: P proves to V the knowledge of witness w for statement z such that $\mathbf{R}'(z,w) = 1$. Here, $z = (x, \overline{c}_1, \overline{c}_2)$, and for $w = (w_1, \sigma, \omega, j, i)$ – such that $j \in \{1,2\}, i \in [\kappa]$ – the relation $\mathbf{R}'(z,w) = 1$ if and only if one of following two holds,

(a) $R_L(x, w_1) = 1$

(b) $c_{j,i} = \text{COM}_{\text{PB}}(\sigma; \omega)$, where $c_{j,i} \in \overline{c}_1 \cup \overline{c}_2$.

This is done by P and V executing the 3-round *statistical* witness indistinguishable (SWI) argument of knowledge obtained by κ parallel repetitions of the Blum-Hamiltonicity (BH) protocol. P uses $(\text{COM}_{\text{PH}}, \text{DCOM}_{\text{PH}})$, the common input is $f(z)$, and P's private input is $g(z,w)$.

Figure 2: Protocol $\pi_{\text{SZA}} : \langle P, V \rangle$.

Let ν be a random variable denoting the joint view of A and V in a real execution of π_{SE}. Similarly, $\nu^{(i)}$ will be the random variable denoting the output of hybrid simulator $\mathcal{H}_i$, $i = 0, 1, 2$. Our final simulator, S, will be the last hybrid simulator, $\mathcal{H}_2$, in the series.

Simulator $\mathcal{H}_0$. This simulator is provided with prover's auxiliary input w (a witness for the common input x). The simulator starts interacting with $A(x, z)$, where z is A's auxiliary input. On left, $\mathcal{H}_0$ acts as the honest prover P using input w (and uniform random tape). On right, $\mathcal{H}_0$ acts as the honest verifier V (with uniform random tape). When A halts, $\mathcal{H}_0$ outputs the (joint) view of A and V, denoted $\nu^{(0)}$, and halts. The simulation is perfect, and we have that,

$$\nu \equiv \nu^{(0)}$$

Simulator $\mathcal{H}_1$. This simulator is identical to $\mathcal{H}_0$ except that it also extracts the witness $\overline{a}$ (using K_{SZA})[4]. If extraction fails, the simulator aborts. Formally, $\mathcal{H}_1$ proceeds exactly as $\mathcal{H}_0$ up until the point where π_{SZA} finishes on left. Let st denote the state of $\mathcal{H}_1$ at this point. Further, let μ denote the view of the verifier (of π_{SZA}) in state st. Let P^*_{SZA} be the machine which is identical to $\mathcal{H}_0$ (up to the point where π_{SZA} finishes on left) except that during the execution of π_{SZA} it interacts with an external verifier V_{SZA} instead of emulating it internally. P^*_{SZA} halts as soon as π_{SZA} finishes on left. If μ is accepting, the simulator $\mathcal{H}_1$ proceeds just like K_{SZA}, with input view μ and oracle access to the machine P^*_{SZA}, to extract a value $\overline{a}$. If K_{SZA} aborts, $\mathcal{H}_1$

[4] K_{SZA} is a standard type of extractor, discussed at the end of this section.

(Slot 1)

$P \Leftrightarrow V$: P proves to V the knowledge of C such that $R_H(G, C) = 1$, using the 3-round (negligible soundness error) BH-protocol.

(Slot 2)

$P \Leftrightarrow V$: P proves to V the knowledge of C such that $R_H(G, C) = 1$, using the 3-round (negligible soundness error) BH-protocol.

Figure 3: Protocol $\pi_{\text{SWA}} : \langle P, V \rangle$.

aborts the simulation. Otherwise, $\mathcal{H}_1$ stores the extracted value $\overline{a}$, and continues the simulation from state st exactly as $\mathcal{H}_0$.

From the description of K_{SZA} it follows that except with negligible probability (in κ), the value recorded by $\mathcal{H}_1$ is $\overline{a}$ such that $R_{\overline{\text{DL}}}(\overline{\text{DL}}, \overline{a}) = 1$. Extraction takes expected polynomial time. Further, if extraction does not fail, outputs of $\mathcal{H}_1$ and $\mathcal{H}_0$ would be identically distributed. As extraction fails only with negligible probability, we have that,

$$\nu^{(0)} \equiv_s \nu^{(1)}$$

Simulator $\mathcal{H}_2$. This simulator is identical to $\mathcal{H}_1$ except that it uses witness $\overline{a}$ (instead of w) in order to execute the protocol π_{SWA}. That is, in the last but one step, for statement $z = (x, \overline{\text{DL}})$, $\mathcal{H}_2$ uses $w' = (\perp, \overline{a})$ to obtain a hamiltonian cycle $g(z, w')$ in graph $f(z)$, which is then used to execute π_{SWA}. Other than this, $\mathcal{H}_2$ and $\mathcal{H}_1$ are identical. We claim that,

$$\nu^{(1)} \equiv_s \nu^{(2)} \tag{1}$$

This claim follows from statistical witness-indistinguishability (SWI) of π_{SWA}. If not, we contradict SWI of π_{SWA} as follows. The cheating verifier, V^*, acts identically to the machine $\mathcal{H}_2$ except that instead of internally emulating the actions of the prover P_{SWA} with witness $\overline{a}$, it interacts with an external prover (who uses either w or $\overline{a}$). At the end of protocol execution V^* outputs its view and halts. If the external prover uses w (resp., $\overline{a}$), the output of V^* is distributed identically to $\nu^{(1)}$ (resp., $\nu^{(2)}$). Thus, if $\nu^{(1)}$ and $\nu^{(2)}$ are not statistically indistinguishable, it will contradict SWI of π_{SWA}.

Our final simulator S is $\mathcal{H}_2$. Our extractor, K, is described next. As discussed in the overview, even though a very simple extraction procedure is possible, we choose to describe a slightly different extraction procedure (the *2*-slot extraction procedure). This would be helpful to us later. We start by presenting some notation.

During simulation, S interacts with $A(x, z)$, simulating interactions on left and right with A controlling the scheduling of the messages. In a view, ν_S, output by the simulator S, based on the appearance of the left hand side tag, VK, we define two events. For this purpose, let the messages of π_{SWA} be denoted by $(\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \gamma_2)$ where $(\alpha_i, \beta_i, \gamma_i)$ are the three messages of the BH-protocol in slot-i, with β_i being V's challenges and $i \in \{1, 2\}$. By our notation, the messages of π_{SWA} in the corresponding *right* hand side session are denoted by $(\tilde{\alpha}_1, \tilde{\beta}_1, \tilde{\gamma}_1, \tilde{\alpha}_2, \tilde{\beta}_2, \tilde{\gamma}_2)$. A view ν_S is said to be of "type-2" if in this view adversary A schedules the delivery of (left) tag VK *before* the delivery of $\tilde{\beta}_2$. Similarly, it is said to be of "type-1" if in this view VK appears *after* $\tilde{\beta}_2$. Call a view ν_S "accepting" if V's view (i.e., the right hand side interaction) in ν_S is accepting.

Intuitively, views of type-2 are those in which slot-2 of π_{SWA} is "free" for K to rewind A and perform extraction. This is because VK appears before $\tilde{\beta}_2$, and hence rewinding A in this slot does not change VK across various threads of execution. Similarly, in type-1, slot-1 is "sort of free". This is because if VK appears in this execution after $\tilde{\beta}_2$, then within "reasonable" amount of trials, K will see another

accepting such accepting conversation. Thus, it can perform rewinding in slot-1, treating VK as a spoiling (or $\perp$) message. A formal description of K appears in figure 4.

Extractor K. On input a view, ν_S, if the right execution in ν_S is aborting, K outputs $\perp$. Otherwise, let the messages of right π_{SWA} in ν_S be $(\tilde{\alpha}_1, \tilde{\beta}_1, \tilde{\gamma}_1, \tilde{\alpha}_2, \tilde{\beta}_2, \tilde{\gamma}_2)$.

1. If VK appears before $\tilde{\beta}_2$ then,

   Define the machine P_2^* as follows. P_2^* is the simulator S (or $\mathcal{H}_2$) initialized with the view ν_S up to the point where S is about to send $\tilde{\beta}_2$ to the internal adversary A. Note that message $\tilde{\alpha}_2$ has already been sent and VK has also been delivered. On input a random challenge $\tilde{\beta}_2'$ (on behalf of V_{SWA}), P_2^* acts as follows. It first forwards $\tilde{\beta}_2'$ to the internal adversary A. P_2^* then continues to interact with A internally exactly as S would (with fresh randomness for the rest of the execution). If A outputs a valid response $\tilde{\gamma}_2'$ (for slot-2 of π_{SWA}) P_2^* outputs $\tilde{\gamma}_2'$ and halts. Otherwise, if A aborts, P_2^* also aborts the execution and halts. Using P_2^*, K now performs extraction as follows:

   (a) Uniformly sample a random $\tilde{\beta}_2'$. If $\tilde{\beta}_2' = \tilde{\beta}_2$, output $\perp$ and halt. Otherwise, feed $\tilde{\beta}_2'$ to P_2^*. If a valid $\tilde{\gamma}_2'$ is received, compute a witness w using $(\tilde{\alpha}_2, \tilde{\beta}_2, \tilde{\gamma}_2, \tilde{\beta}_2', \tilde{\gamma}_2')$ (and function g^{-1}) and halt. Otherwise, repeat this step with a new random challenge.

2. If VK appears after $\tilde{\beta}_2$ then,

   Define the machine P_1^* as follows. P_1^* is the simulator S (or $\mathcal{H}_2$) initialized with the view ν_S up to the point where S is about to send $\tilde{\beta}_1$ to the internal adversary A. Note that message $\tilde{\alpha}_1$ has already been sent *but* VK has not yet appeared in the conversation. On input a random challenge $\tilde{\beta}_1'$ (on behalf of V_{SWA}), P_1^* acts as follows. It first forwards $\tilde{\beta}_1'$ to the internal adversary A. P_1^* then continues to interact with A internally exactly as S would (with fresh randomness for the rest of the execution). At some point, P_1^* either delivers the tag VK or outputs a valid response $\tilde{\gamma}_1'$ (for slot-1 of π_{SWA}). If VK is scheduled, P_1^* outputs $\perp$ and halts. Otherwise (i.e., A outputs $\tilde{\gamma}_1'$), P_1^* outputs $\tilde{\gamma}_1'$ and halts. Using P_1^*, K now performs extraction as in (1a):

   (a) Uniformly sample a random $\tilde{\beta}_1'$. If $\tilde{\beta}_1' = \tilde{\beta}_1$, output $\perp$ and halt. Otherwise, feed $\tilde{\beta}_1'$ to P_1^*. If a valid $\tilde{\gamma}_1'$ is received, compute a witness w using $(\tilde{\alpha}_1, \tilde{\beta}_1, \tilde{\gamma}_1, \tilde{\beta}_1', \tilde{\gamma}_1')$ (and function g^{-1}) and halt. Otherwise, repeat this step with a new random challenge.

Figure 4: Extractor K

We have the following lemma.

Lemma 4 *K runs in expected polynomial time.*

Proof. If ν_S is aborting, K halts, taking $\text{poly}(\kappa)$ steps. Otherwise, it executes either step (1) or (2). We show that each of these steps runs in expected polynomial time. First, consider step (1). Let p_1 denote the probability that P_2^* outputs a valid answer, where the probability is taken over the randomness needed for both: creating $\tilde{\beta}_2$ and internally interacting with A. Then, because ν_S is accepting, step (1a) gets executed with probability at most p_1. Further, expected number of steps taken by K in (1a) are $\text{poly}(\kappa)/p_1$. As K invests at most $\text{poly}(\kappa)$ steps in constructing P_2^*, the expected number of steps taken by K in step (1) are at most $\text{poly}(\kappa) + p_1 \cdot \frac{\text{poly}(\kappa)}{p_1} = \text{poly}(\kappa)$.

To compute the steps invested by K in step (2), let p_2 denote the probability that P_1^* outputs a valid response, where the probability is taken over the randomness needed for both: creating $\tilde{\beta}_1$ and internally interacting with A. Expected number of steps taken by K in (2) are $\text{poly}(\kappa)/p_2$. The key observation is that step (2) gets executed with probability at most p_2. Indeed, the probability of executing step (2) is equal

to the probability that in an (accepting view) ν_S, tag VK appears after $\tilde{\beta}_2$. This is less than or equal to the probability that VK appears after the valid (slot-1) response $\tilde{\gamma}'_1$. But this probability is p_2, because P_1^* outputs $\perp$ whenever VK is delivered before $\tilde{\gamma}'_1$. Hence, expected number of steps invested by K in step (2) are at most $\text{poly}(\kappa) + p_2 \cdot \frac{\text{poly}(\kappa)}{p_2} = \text{poly}(\kappa)$. This concludes the proof. ∎

Note that K uses the simulator S while performing extraction. Ability to simulate while extracting simultaneously, is what allows the extraction to succeed. We will use the phrase "K_S outputs $\tilde{w}$" to insist that K uses the simulator S to extract the witness. This notation will be convenient in future.

Lemma 5 *If ν_S is accepting in the right hand side interaction, let $\tilde{x}$ be the theorem proven by $A(x, z)$ to V. Then, except with negligible probability in κ, K outputs $\tilde{w}$ such that $R_L(\tilde{x}, \tilde{w}) = 1$.*

Proof. K outputs either $\perp$ or a w extracted due to the special-soundness property of the BH-protocol. The output is $\perp$ only when the challenge $\tilde{\beta}_2$ (or $\tilde{\beta}_1$) appearing in view ν_S is repeated. As $|\tilde{\beta}_i| = \kappa$ (for $i \in \{1, 2\}$), challenges are sampled uniformly at random, and K runs in expected polynomial time, we conclude that this happens with only negligible probability. Hence, K outputs $\tilde{w}$ such that $\mathbf{R}(\tilde{z}, \tilde{w}) = 1$ where $\tilde{z} = (\tilde{x}, \widetilde{\overline{\text{DL}}})$. For notational ease, let $\widetilde{\text{DL}}$ denote $\widetilde{\overline{\text{DL}}}$. By definition of $\mathbf{R}$, it follows that $\tilde{w}$ is such that either $R_L(\tilde{x}, \tilde{w}) = 1$ or $R_{\overline{\text{DL}}}(\widetilde{\text{DL}}, \tilde{w}) = 1$. Lemma 6 shows that the second case is highly unlikely. Hence the lemma. ∎

Notation. Let BAD denote the event that "K_S outputs $\tilde{w}$, such that $R_{\overline{\text{DL}}}(\widetilde{\text{DL}}, \tilde{w}) = 1$". Later on, we will introduce various hybrid simulators S_i, $i \in [3]$, by slightly modifying S. When K works with simulator S_i to perform extraction, we define event BAD_i (analogous to BAD) to be the following: "K_{S_i} outputs $\tilde{w}$, such that $R_{\overline{\text{DL}}}(\widetilde{\text{DL}}, \tilde{w}) = 1$".

Lemma 6 *If* GAP-DL *holds, then* $\Pr[\mathit{BAD}] \leq \text{neg}(\kappa)$

Before proceeding to the proof of this lemma, observe that equation (1) along with lemmata 4 and 5, implies the theorem of this section:

Theorem 7 *Assume that* GAP-DL *holds, and that perfectly hiding commitments exist. Protocol $\pi_{\text{SE}} : \langle P, V \rangle$ is then a simulation-extractable argument for all languages in* NP.

The rest of this section is now devoted to proving lemma 6.

Proof of Lemma 6. Assume that the lemma does not hold. Then, K outputs $\tilde{w}$ – which is a solution to the instance $\widetilde{\text{DL}}$ – with non-negligible probability, say ϵ. The probability is taken over the random tape of the simulator-extractor $\text{SE} = (S, K)$. We show how to construct a machine for breaking the GAP-DL. We have that $\overline{\text{DL}} = (\text{DL}_1, \ldots, \text{DL}_\kappa), \widetilde{\overline{\text{DL}}} = (\widetilde{\text{DL}}_1, \ldots, \widetilde{\text{DL}}_\kappa), \text{DL}_i = (g_i, y_i, p_i), \widetilde{\text{DL}}_i = (\tilde{g}_i, \tilde{y}_i, \tilde{p}_i)$. Note that Because A does not copy the tag, from claim 3 we have that there exists a target prime $\tilde{p}_i$ which is "far" from all p_j. (Note that the condition $|p| \neq |q|$ for two primes p, q implies that $p \neq q$, because the two primes lie in two disjoint intervals of the number line). Let us call the DL-instance $\widetilde{\text{DL}}_i$ associated with the target prime $\tilde{p}_i$, the *target instance*. There may be multiple indices i for which claim 3 holds. In such a case, one of them is chosen arbitrarily to be named as the target index. For notational ease, let $\tilde{p} \stackrel{\text{def}}{=} \tilde{p}_i$.

To construct a machine for breaking GAP-DL we now proceed gradually by introducing some hybrid machines. First, for convenience, we slightly modify the query structure of $\mathcal{O}_{p,\ell(\kappa)}$ to accept an additional string st of length $\text{poly}(\kappa)$. Additionally, instead of serving one query at a time, the oracle can receive multiple queries at the same time (in the form of a vector), and returns a vector of answers. That is, a query to $\mathcal{O}_{p,\ell(\kappa)}$ is of the form $(\overline{\text{DL}}, \mathsf{st})$, where $\overline{\text{DL}} = (\text{DL}_1, \ldots, \text{DL}_m)$. The oracle ignores the string st, and computes an answer a_i to each DL_i as described in section 3. Finally, it returns $\overline{a} = (a_1 \ldots, a_m)$.

Simulator $S_1^{\mathcal{O}_{\tilde{p},\ell(\kappa)}}$. For notational ease, let $S_1 \stackrel{\text{def}}{=} S_1^{\mathcal{O}_{\tilde{p},\ell(\kappa)}}$. Simulator S_1 is *identical* to S except that instead of executing the knowledge-extractor K_{SZA} when π_{SZA} (on left) finishes, it constructs the witness $\overline{a}$ as follows. Let $\overline{\text{DL}} = (\text{DL}_1, \ldots, \text{DL}_\kappa)$ be the common input of π_{SZA}. S_1 sends a query $(\overline{\text{DL}}, \mathsf{st})$ to oracle $\mathcal{O}_{\tilde{p},\ell(\kappa)}$ and receives an answer $(a_1 \ldots, a_\kappa)$. Witness $\overline{a}$ is set to be the vector $(a_1 \ldots, a_\kappa)$. If $\overline{a}$ is such that $R_{\overline{\text{DL}}}(\overline{\text{DL}}, \overline{a}) = 1$, S_1 continues with the simulation. Otherwise, it aborts. We claim the following,

Claim 8 $\Pr[\mathit{BAD}_1] \geq \epsilon$

Proof. First, observe that because of claim 3, it follows that the query sent by S_1 to $\mathcal{O}_{\tilde{p},\ell(\kappa)}$ respect the conditions set-forth by GAP-DL (in the modified description). That is, each component of $\overline{\text{DL}}$ is a DL-instance with a prime of proper size. Next, we would like to show that machines S_1 and S are *identical*[5]. To do this, we look at the following alternative descriptions of S.

Alternative description of S. Simulator S runs the knowledge-extractor K_{SZA} (guaranteed for π_{SZA}) as a subroutine. Instead of looking at K_{SZA} as a subroutine, we consider K_{SZA} as an oracle which S is given access to. Formally, S acts as $\mathcal{H}_1$ upto the point where π_{SZA} finishes on left. Now, S sends query $(\overline{\text{DL}}, \mathsf{st})$ to oracle K_{SZA}. Here $\overline{\text{DL}}$ is the common input of (left execution of) π_{SZA} and st is the state of the simulator (which includes the state of the internal adversary A) upto the point where (left) π_{SZA} finishes. K_{SZA} returns $\overline{a}$ (which could also be $\perp$). S then proceeds exactly as $\mathcal{H}_2$ (using $\overline{a}$) from this point onwards.

From the alternative description of S, we deduce that the only difference between S_1 and S is that they have access to different oracles. However, it does not affect the binary representation of the two machines. Hence, we conclude that S_1 is identical to S. This, however, is not enough because computation of $\tilde{w}$ using K uses the respective oracles of S and S_1. Hence, we need to show that whenever K_S outputs $\tilde{w}$ (and not $\perp$), computation by K_{S_1} proceeds *identically* to the computation by K_S. Thus whenever, K_S outputs $\tilde{w}$, so will K_{S_1}. Let $(\overline{q}_1, \overline{q}_2, \ldots, \overline{q}_i, \ldots)$ be the queries made by K_S to K_{SZA}, and $(\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_i, \ldots)$ be the respective answers received. As K_S outputs $\tilde{w}$, it holds that $\forall i, \overline{a}_i \neq \perp$. Analogously, let $(\overline{Q}_1, \overline{Q}_2, \ldots, \overline{Q}_i, \ldots)$ and $(\overline{A}_1, \overline{A}_2, \ldots, \overline{A}_i, \ldots)$ be the queries (to $\mathcal{O}_{\tilde{p},\ell(\kappa)}$) and answers (received) during the computation by K_{S_1}. To prove our claim, we need to show that $\forall i$: $\overline{Q}_i = \overline{q}_i$ and $\overline{A}_i = \overline{a}_i | \overline{a}_i \neq \perp$. We demonstrate this by induction. First of all, as S and S_1 are identical, we have that $\overline{Q}_1 = \overline{q}_1$. Note that if answer $\overline{a}_1$ (returned by K_{SZA} to the query $\overline{q}_1$) is not $\perp$, it is a solution to the DL-instance in $\overline{q}_1$. On the other hand, answer $\overline{A}_1$ (returned by $\mathcal{O}_{\tilde{p},\ell(\kappa)}$ to the query $\overline{Q}_1$) is also a solution to the DL-instance in $\overline{Q}_1$, *if $\overline{Q}_1$ is a valid query.* As shown below, because of the 2-slot extraction mechanism, all queries made by K_{S_1} to $\mathcal{O}_{\tilde{p},\ell(\kappa)}$ are valid. Because, $\overline{Q}_1 = \overline{q}_1$ and each DL-instance has a *unique* solution, it holds that $\overline{A}_1 = \overline{a}_1 | \overline{a}_1 \neq \perp$. By applying the *same* argument it can be concluded that $\overline{Q}_i = \overline{q}_i$ and $\overline{A}_i = \overline{a}_i | \overline{a}_i \neq \perp$ for all i. We finish the proof by showing that each $\overline{Q}_i$ is a valid query.

Every $\overline{Q}_i$ is a valid query. Let $\text{VK}, \widetilde{\text{VK}}$ be the left and right tags respectively, in the view ν_S output by S_1. If the view is of "type-2", then both $\text{VK}, \widetilde{\text{VK}}$ have already appeared in the partial view used to construct P_2^* (see the description of K). However, the common input $\overline{\text{DL}}$ for the left execution of π_{SZA} may not yet have been decided. But because $\text{VK}, \widetilde{\text{VK}}$ are fixed and the same for all execution paths explored by K, every common input $\overline{\text{DL}}$ to (left) π_{SZA} in these execution paths satisfies the conditions setforth by the GAP-DL. That is, for every $\text{DL}_i \in \overline{\text{DL}}$ in every execution path explored by K, it holds that $\text{DL}_i = (p_i, g_i, y_i)$ is a DL-instance and $\big||p_i| - |\tilde{p}|\big| \geq \ell(\kappa)$ where $\tilde{p}$ is the target prime. Hence, all queries made by K_{S_1} in this case are valid. In the second case, when ν_S is of "type-1", note that P_1^* aborts whenever VK appears in the path

[5]Every Turing machine M can be written down as a binary string, denoted $\mathsf{bin}(M)$. Turing machines M_1, M_2 are said to be identical if and only if $\mathsf{bin}(M_1) = \mathsf{bin}(M_2)$.

being explored. Hence, in this case, K_{S_1} *never* makes any queries to $\mathcal{O}_{\tilde{p},\ell(\kappa)}$. This concludes the argument.

As computations by K_S and K_{S_1} proceed identically whenever K_S does not abort, we conclude that K_{S_1} outputs $\tilde{w}$ such that $R_{\overline{\text{DL}}}(\widetilde{\text{DL}}, \tilde{w}) = 1$ with probability at least ϵ. ∎

Note that K_{S_1} runs in *strict* polynomial time. In order to keep things simple, we present what seems to be the simplest (yet formal) description of our next two hybrid simulators. This helps us in keeping things simple without going into (tedious) formal details. Nevertheless, these formal and rigorous descriptions of S_2 and S_3 are necessary for the claims proven about these simulators, and hence they are provided in appendix A. If one prefers a rigorous formalization, the descriptions given in appendix A should be used.

The final step is to modify S_1 so that instead of internally emulating the proof using π_{SZA}, it uses the *2*-slot simulator, S_{SZA}, provided later (in figure 5). Instead of directly performing this step, we do it in two stages. In the first step, we only extract the *trapdoor* witness (of the FLS-style trick) but do not use it (see simulator S_2). In the next and final step, we use this trapdoor witness instead of the actual witness $\overline{a}$ for $\overline{\text{DL}}$.

Simulator S_2. This simulator is identical to S_1 (with access to $\mathcal{O}_{\tilde{p},\ell(\kappa)}$) except for the following difference. When "slot-2" of π_{SZA} *on right* finishes, S_2 executes the (2-slot) extractor procedure, `EXTRACT` (with VK acting as the "special" message `sp`), to extract a witness $w_2 = (\sigma, \omega, i, j)$ for the intermediate statement $\overline{\text{DL}}, \tilde{c}_1, \tilde{c}_2)$.[6] This is done by constructing a machine V_1^*, which is identical to S_1 except that it does not emulate the part of V which executes π_{SZA} (on right). Instead, V_1^* expects it to be an external party. V_1^* halts as soon as "slot-2" of (right) π_{SZA} finishes. Procedure `EXTRACT` now extracts w_2 from V_1^*. (The procedure `EXTRACT` is a part of the 2-slot simulator for π_{SZA}.) If `EXTRACT` fails (i.e., $w_2 = \perp$), S_2 aborts. Otherwise, S_2 continues exactly as S_1 (with the actual witness $\tilde{a}$ for $\widetilde{\text{DL}}$).

Claim 9 $\Pr[\mathit{BAD}_2] \geq \epsilon/2$

Proof. First, note that when `EXTRACT` computes w_2, it rewinds the internal adversary A in the right execution. As a result, the left execution is also rewound. If the left execution is rewound past the point where A outputs the (left) statement $\overline{\text{DL}}$ (for π_{SZA}), S_2 may later make a new query to $\mathcal{O}_{\tilde{p},\ell(\kappa)}$ when the execution reaches the end of left π_{SZA}. We need to argue that these new queries are valid, so that the execution will continue exactly as in S_1, and hence `EXTRACT` will be able to compute w_2. As `EXTRACT` is a 2-slot procedure (similar to K), using the same arguments as in the proof of claim 8, it can be (easily) deduced that all new queries are indeed valid. (Details are repetitive and omitted). Next, note that if `EXTRACT` does not output $\perp$, the computation in the "main thread" for both K_{S_1} and K_{S_2} are identical. Hence, K_{S_2} also outputs $\tilde{w}$ such that $R_{\overline{\text{DL}}}(\widetilde{\text{DL}}, \tilde{w}) = 1$ (provided, `EXTRACT` does not abort). As `EXTRACT` aborts with only negligible probability, we have that $\Pr[\texttt{BAD}_2] \geq \epsilon(1 - \text{neg}(\kappa)) \geq \epsilon/2$. Because `EXTRACT` runs in expected polynomial time, the running time of K_{S_2} is also expected polynomial. ∎

Note that in the proof above, we have used the term "main thread". This is not needed if one is using the rigorous formalization provided in appendix A (because we directly deal with the actual machine and its binary representation, as done in claim 8).

Simulator S_3. This simulator is identical to S_2 (with access to $\mathcal{O}_{\tilde{p},\ell(\kappa)}$) except for the following difference. Recall that the SWI argument-of-knowledge part of the right π_{SZA} (on common input $\widetilde{\text{DL}}$) is executed on an intermediate statement $(\widetilde{\text{DL}}, \tilde{c}_1, \tilde{c}_2)$. S_3 uses witness w_2 (obtained using the `EXTRACT` procedure), instead of $\tilde{a}$, to execute this SWI argument-of-knowledge.

[6] For notational ease, we are using $\tilde{c}_i$ instead of $\widetilde{\tilde{c}}_i$. The scheduling of "special" symbol affects the extraction, see description of `EXTRACT`.

Claim 10 $\Pr[\mathit{BAD}_3] \geq \epsilon/4$

Proof. We prove this claim by constructing an exponential time adversary $\mathcal{A}_{\text{SWI}}$ for the SWI property of the BH-protocol as follows. $\mathcal{A}_{\text{SWI}}$ proceeds as follows:

1. It internally proceeds exactly as S_2, except that it does not emulate the part of V which acts as the prover of BH-protocol for the statement $(\widetilde{\text{DL}}, \tilde{c}_1, \tilde{c}_2)$. Instead, it expects this proof to be coming from an outside prover, say P_{BH}, who is either using $\tilde{a}$ or w_2, and $\mathcal{A}_{\text{SWI}}$ is acting as the BH-verifier for P_{BH}. During the internal simulation, at some point interaction with P_{BH} also finishes. $\mathcal{A}_{\text{SWI}}$ then continues internally to obtain a view, say ν_S.

2. Simulator S_2 needs access to oracle $\mathcal{O}_{\tilde{p},\ell(\kappa)}$. $\mathcal{A}_{\text{SWI}}$ internally simulates this oracle by computing answers to the valid queries of S_2 in exponential time. (This is the only reason why $\mathcal{A}_{\text{SWI}}$ is exponential time)

3. $\mathcal{A}_{\text{SWI}}$ now runs extractor K on input view ν_S. Note that K works by constructing P_1^* and P_2^*, and exploring various execution paths for right π_{SWA} (left executions are internally simulated by P_1^* or P_2^*). External P_{BH} is thus not required to execute this step. K outputs a value $\tilde{w}$. If $R_{\overline{\text{DL}}}(\widetilde{\text{DL}}, \tilde{w}) = 1$, $\mathcal{A}_{\text{SWI}}$ outputs 1 and 0 otherwise.

If P_{BH} uses $\tilde{a}$, K's output is distributed like K_{S_2}, and when P_{BH} uses $\tilde{w}$ it is distributed like K_{S_3}. Hence, $\mathcal{A}_{\text{SWI}}$ compromises SWI property with advantage $\left|p_2 - p_3\right|$ where $p_i = \Pr[\mathtt{BAD}_i]$, for $i \in \{2, 3\}$. It thus follows $p_3 \geq p_2 - \text{neg}(\kappa) \geq \epsilon/2 - \text{neg}(\kappa) \geq \epsilon/4$. ∎

Finally, we present our adversary $\mathcal{A}_{\text{GAP-DL}}$ for the GAP-DL assumption. Adversary $\mathcal{A}_{\text{GAP-DL}}$ acts as follows,

1. It samples an index $\mathsf{t} \in [\kappa]$ uniformly at random, as its guess of the target index and a bit b as its guess of bit $\widetilde{\text{VK}}_{\mathsf{t}}$. Let $\ell'(\kappa) = (\tilde{t}_{\mathsf{t}} + 1) \cdot \ell(\kappa)$, where $\tilde{t}_{\mathsf{t}} = \mathsf{t} \circ \widetilde{\text{VK}}_{\mathsf{t}} = \mathsf{t} \circ b$. $\mathcal{A}_{\text{GAP-DL}}$ sends $\ell'(\kappa)$ to the *challenger*, who responds with a *challenge* DL-instance $\mathsf{ch} = (\tilde{p}, \tilde{g}, \tilde{y})$, such that $|\tilde{p}| = \ell'(\kappa)$.

2. $\mathcal{A}_{\text{GAP-DL}}$ now proceeds exactly as simulator S_3 except for the following two differences:

   (a) As soon as, both VK and $\widetilde{\text{VK}}$ are fixed, the target index i and the bit $\widetilde{\text{VK}}_i$ are also fixed. If $i = \mathsf{t}$ *and* $\widetilde{\text{VK}}_i = b$, then $\mathcal{A}_{\text{GAP-DL}}$ continues the execution as S_3 and aborts otherwise.

   (b) While computing the common input $\widetilde{\text{DL}}$ for the right π_{SZA}, $\mathcal{A}_{\text{GAP-DL}}$ sets the target DL-instance to be the challenge instance. That is, $\widetilde{\text{DL}}_i = \mathsf{ch}$.

3. When simulation finishes, a view – say ν_S – is obtained. $\mathcal{A}_{\text{GAP-DL}}$ now applies the extractor K with input ν_S to the internal simulator S_3 to obtain $\tilde{w}$. Note that $\mathcal{A}_{\text{GAP-DL}}$ is given access to the oracle $\mathcal{O}_{\tilde{p},\ell(\kappa)}$ which it uses to run the (internal) simulator S_3. $\mathcal{A}_{\text{GAP-DL}}$ outputs whatever K outputs.

We claim that $\mathcal{A}_{\text{GAP-DL}}$ solves ch with probability at least $\frac{\epsilon/4}{2\kappa} = \frac{\epsilon}{8\kappa}$. To see this, observe that with probability at least $\frac{1}{2\kappa}$, $\mathcal{A}_{\text{GAP-DL}}$ correctly guesses *both* – the target index i and the bit $\widetilde{\text{VK}}_i$. Given that these guesses are correct, the internal simulation is *identical* to S_3 and hence extraction proceeds as in K_{S_3}. Thus, from claim 10, we obtain a solution to $\widetilde{\text{DL}}$ (which includes a solution to ch) with probability at least $\epsilon/4$. Combining these facts, we conclude that $\mathcal{A}_{\text{GAP-DL}}$ solves ch with probability at least $\frac{\epsilon}{8\kappa}$. If ϵ is non-negligible, it contradicts the GAP-DL. This concludes the proof of lemma 6. ∎

The 2-slot Simulator for π_{SZA}. The protocol π_{SZA} is used inside the larger protocol π_{SE} (which executes in the presence of one of its executions). To argue security of π_{SE}, we need to replace the prover of π_{SZA} with the simulator, S_{SZA}. Thus S_{SZA} is designed by keeping in mind that some of the messages of the (external) protocol π_{SE} may need special care when S_{SZA} rewinds its adversary, V^*. This is captured by allowing V^* to schedule a "special" message, denoted sp. If sp occurs in/before slot-1, S_{SZA} will not rewind V^* past sp.

Further, if sp appears after slot-1 finishes, every "rewind thread" will be immediately stopped if V^* outputs sp in this thread.

Let (r_1, a_1, r_2, a_2) denote the four messages of the two slots of π_{SZA}. That is r_i is prover's challenge to which V^* responds with a_i in slot-i, $i \in \{1, 2\}$. Let $V^*_{r_i}$ denote the (state of) verifier V^* where it expects to receive r_i from the prover. Using this notation, the description of S_{SZA} appears in figure 5.

Simulator S_{SZA}. Honestly execute the protocol π_{SZA} with V^* up to the point where a_2 is received (slot-2 finishes). If V^* aborts before reaching this point, output the view and halt. Otherwise, let ν_1 denote the (partial) view so far. Run the procedure EXTRACT on input view ν_1. If EXTRACT aborts, abort the simulation. Otherwise let $w_2 = (\sigma, \omega, i, j)$ be the witness output by EXTRACT for the intermediate language $(x, \overline{c}_1, \overline{c}_2)$. Continue the execution of π_{SZA} from view ν_1 and use the witness w_2 to complete the 3-round BH-protocol.

Procedure EXTRACT**.** On input a view ν_1, the procedure proceeds as follows.

1. If sp appears before r_2, rewind V^* to the state $V^*_{r_2}$ and do the following.
   Uniformly sample a random r'_2. If $r'_2 = r_2$, output $\perp$ and halt. Otherwise, feed r'_2 to $V^*_{r_2}$. If V^* outputs $\perp$, stop this thread of execution, and repeat the step with a new random challenge. Otherwise, let a'_2 be the valid response received. Compute a witness w_2 using (r_2, a_2, r'_2, a'_2), output w_2, and halt.
2. If sp appears after r_2, rewind V^* to the state $V^*_{r_1}$. Let $V^{**}_{r_1}$ denote the machine which is identical to $V^*_{r_1}$ except that whenever $V^*_{r_1}$ outputs sp, $V^{**}_{r_1}$ outputs $\perp$. Now proceed as follows.
   Uniformly sample a random r'_1. If $r'_1 = r_1$, output $\perp$ and halt. Otherwise, feed r'_1 to $V^{**}_{r_1}$. If $V^{**}_{r_1}$ outputs $\perp$, stop this thread of execution, and repeat this step with a new random challenge. Otherwise, let a'_1 be the valid response received. Compute a witness w_2 using (r_1, a_1, r'_1, a'_1), output w_2, and halt.

Figure 5: Simulator S_{SZA}

Using a standard argument, it can be shown that if EXTRACT outputs a witness w_2 for the intermediate language $(x, \overline{c}_1, \overline{c}_2)$ and BH-protocol is SWI, the output of S_{SZA} will be computationally indistinguishable from the view of a cheating V^* in a real execution of π_{SZA}. We thus only show that EXTRACT outputs such a w_2 with high probability within expected polynomial time. The proof falls along the lines of proof of Lemma 4. We thus only provide a proof-sketch.

Lemma 11 *Procedure* EXTRACT *outputs a valid witness w_2 in expected polynomial time.*

Proof. It is straightforward to see that if EXTRACT successfully obtains valid answers to a different challenge (r'_1 or r'_2) it obtains a valid opening of one of the 2κ commitments. This constitutes a valid witness. Using the proof of lemma 4, it can be easily derived that EXTRACT runs in expected polynomial time (details are repetitive, and omitted). Hence the Lemma. ∎

Extractor $K_{\mathbf{sza}}$**.** This extractor works as follows. Let ν be the input/simulated view. If ν is accepting, it extracts a witness w for statement x w.h.p, given oracle access to P^*_{SZA}, as follows. Let P^*_{BH} denote the residual prover computed from P^*_{SZA} and view ν. P^*_{BH} is the prover of the BH-protocol for the intermediate statement $(x, \overline{c}_1, \overline{c}_2)$. K_{SZA} applies the extractor of the BH-protocol to P^*_{BH} to compute a witness w for $(x, \overline{c}_1, \overline{c}_2)$. Using standard arguments, it can be shown that this process takes expected polynomial time; and *if the commitment scheme is semantically secure*, w is such that $R_L(x, w) = 1$ with high probability. Such proofs are standard in the zero-knowledge literature, and hence we choose to omit further details.

5 Non-interactive and Non-malleable Commitments

Bit Commitment. The commitment scheme is described in figure 6. The committer, C, on input a bit b, executes $\mathbf{Com}(b)$ to compute and send the commitment value, c, to R.

The receiver, R, on receiving the value $c = (\text{VK}, m, \sigma)$, *accepts the commitment* if and only if all of the following tests succeed (here $m = \vec{p} \circ \vec{g} \circ \vec{y} \circ c'$): (1) $\text{VERIFY}(m, \sigma, \text{VK}) = 1$, (2) for all $i \in [\kappa]$, $p_i = 2q_i + \in \vec{p}$ is a safe prime of length $(t_i + 1) \cdot \ell(\kappa)$ where $t_i = i \circ \text{VK}_i$, $g_i \in \vec{g}$ generates the subgroup G_{q_i} of $\mathbb{Z}^*_{p_i}$, $y_i \in \vec{y}$ is an element of G_{q_i}, and c' is a bit. Otherwise it outputs $\perp$, denoting "failure".

To decommit, C sends $(x_1, \ldots, x_\kappa)$ to R, who computes the committed bit as follows. First, it tests that $\forall i, y_i = f_i(x_i)$. If so, it sets the committed bit b to be $c' \oplus (\oplus_{i=1}^{\kappa} \text{HCB}_{f_i}(x_i))$. Otherwise, it outputs $\perp$, denoting failure.

Commitment: Com(b).

Step 1: Initialize functions
Compute $(\text{VK}, \text{SK}) \leftarrow \mathcal{K}(1^\kappa)$ and $(p_1, \ldots, p_\kappa) \leftarrow \text{PRIMES}(\text{VK})$.
For all $i \in [\kappa]$, define g_i to be a random generator of $G_{q_i} \subset \mathbb{Z}^*_{p_i}$.
Let $\vec{p} = (p_1, \ldots, p_\kappa)$ and $\vec{g} = (g_1, \ldots, g_\kappa)$.
Define $f_i(x) = {g_i}^x \mod p_i$.

Step 2: Create the commitment
Choose $x_1, \ldots, x_\kappa$ independently at random and set,

$$c' = b \oplus (\oplus_{i=1}^{\kappa} \text{HCB}_{f_i}(x_i))$$

Let $\vec{y} = (y_1, \ldots, y_\kappa)$ where $y_i = f_i(x_i)$.
Let $m = \vec{p} \circ \vec{g} \circ \vec{y} \circ c'$; compute $\sigma \leftarrow \text{SIGN}(m, \text{SK}, \text{VK})$.
The commitment value is, $c = (\text{VK}, m, \sigma)$. Send c to R.

Decommitment: Dcom(c).

Send $(x_1, \ldots, x_\kappa)$ to R.

Figure 6: Our NINM Bit-commitment Scheme

The following theorem establishes that $(\mathbf{Com}, \mathbf{Dcom})$ is a NINM bit commitment scheme.

Theorem 12 *If* GAP-DL *holds, the scheme* $(\mathbf{Com}, \mathbf{Dcom})$ *is a non-interactive* perfectly binding *and computationally hiding bit-commitment scheme that is* non-malleable with respect to commitment.

Proof. To prove the theorem, we need to demonstrate the following properties: perfect hiding, computational binding, and non-malleability with respect to commitment.

Suppose R accepts c as a valid commitment. Every f_i defined by p_i, g_i, is a *permutation* over $G_{q_i} \subset \mathbb{Z}^*_{p_i}$. Hence every y_i defines a unique preimage x_i, and hence a unique hard-core bit, $\text{HCB}_{f_i}(x_i)$. Perfect binding thus follows.

We now demonstrate that if GAP-DL holds, our commitment scheme is non-malleable with respect to commitment. The man-in-the-middle, A, produces a commitment $\tilde{c}$ on receiving a commitment c. Let $\tilde{b}_0$ be a random variable, denoting the bit committed to by A when c is a commitment to 0. Similarly, let $\tilde{b}_1$ be a random variable, denoting the bit committed to by A when c is a commitment to 1. To show that our commitment scheme is non-malleable with respect to commitment, we only need to argue that

$\tilde{b}_0$ and $\tilde{b}_1$ are computationally indistinguishable. To argue this, assume to the contrary that there exists a PPT distinguisher D such that the following quantity, ϵ, is non-negligible,

$$\epsilon = \left|\Pr[D(\tilde{b}_0) = 1] - \Pr[D(\tilde{b}_1) = 1]\right|$$

We construct an adversary, A^*, against the GAP-DL assumption using A as a black-box. We first generate VK and guess the *target index*, i, at random. The key VK and the targer index i is hardwired into A^* and will remain fixed in all of its invocations. A^* now requests a random DL-instance (p, g, y) such that p has length $(t_i + 1) \cdot \ell(\kappa)$, where $t_i = i \circ \text{VK}_i$. It then proceeds to compute (the solution) x s.t. $y = g^x \mod p$ as follows,

- Generate $\vec{p}, \vec{g}$ exactly as in **Com** except that set $p_i = p$ and $g_i = g$. For all $j \neq i$, select x_j at random, and compute $y_j = f_j(x_j)$ where $f_j \stackrel{\text{def}}{=} {g_j}^{x_j} \mod p_j$. Set $y_i = y$ to define $\vec{y} = (y_1, \ldots, y_\kappa)$. Compute $b_j = \text{HCB}_{f_j}(x_j)$ for all $j \neq i$, and choose c' at random.
- Set $c = (\text{VK}, m, \sigma)$ where $m = \vec{p} \circ \vec{g} \circ \vec{y} \circ c'$ and $\sigma \leftarrow \text{SIGN}(m, \text{SK}, \text{VK})$. By construction, c is an accepting commitment, and hence it defines a unique bit $b = c' \oplus (\oplus_{j=1}^{\kappa} b_j)$; which is not yet known to A^* as b_i – representing the hard-core bit corresponding to the target DL-instance (p_i, g_i, y_i) – is not yet known to A^*.
- Input c to A to obtain a commitment $\tilde{c}$. That is, $\tilde{c} \leftarrow A(c)$. Let $\tilde{c} = (\widetilde{\text{VK}}, \tilde{m}, \tilde{\sigma})$, where $\tilde{m} = (\tilde{p} \circ \tilde{g} \circ \tilde{y} \circ c'')$. Further, let $\tilde{p} = (\tilde{p_1}, \ldots, \tilde{p_\kappa})$, $\tilde{g} = (\tilde{g_1}, \ldots, \tilde{g_\kappa})$, $\tilde{y} = (\tilde{y_1}, \ldots, \tilde{y_\kappa})$.
- If $\tilde{c}$ is not accepting, or $\tilde{c} = c$, or $\widetilde{\text{VK}}_i = \text{VK}_i$, abort. (Recall that i is the target-index). Otherwise, define k DL-instances $\widetilde{\text{DL}}_j = (\tilde{p}_j, \tilde{g}_j, \tilde{y}_j)$, and query $\mathcal{O}_{p_i, \ell(\kappa)}$ for each $\widetilde{\text{DL}}_j$ ($j \in [\kappa]$) to obtain the solution $\tilde{x}_j$ for $\widetilde{\text{DL}}_j$. From $\tilde{x}_j$, compute the hard-core bit $\tilde{b}_j$ for $\widetilde{\text{DL}}_j$. Define the bit $\tilde{b} = c'' \oplus (\oplus_{j=1}^{\kappa} \tilde{b}_j)$. Note that $\tilde{b}$ is the bit to which $\tilde{c}$ is a commitment.
- Let $d \leftarrow D(\tilde{b})$.[7] Compute $d' = d \oplus c' \oplus (\oplus_{j \neq i} b_j)$. Assuming d' to be the hard-core bit b_i for the target instance, A^* computes the solution x_i for DL_i. If x_i is a valid solution, output x_i. Otherwise output $\perp$.

A^* **succeeds.** We now show that A^* indeed solves the DL-instance (p, g, y) using the oracle $\mathcal{O}_{p,\ell(\kappa)}$, with noticeable probability. First, observe that all the DL-instances in c and bit c' are computed uniformly at random. Hence c is a commitment to a *random* bit.

It follows from the definition of ϵ on input c, A outputs an accepting $\tilde{c} \neq c$, with probability at least ϵ. Further, the bit $\tilde{b}$ represented by $\tilde{c}$, is distributed identically to $\tilde{b}_0$ (resp., $\tilde{b}_1$) when c is a commitment to 0 (resp., 1). As the overall probability of A's success is ϵ, using a standard counting argument, it holds that for (at least) an $\epsilon/2$ fraction of keys VK, A succeeds with probability at least $\epsilon' = \epsilon/2$. Fix such a VK that is hardwired into A^*.

From the strong unforgeability of the signature scheme, it holds that if $\tilde{c} \neq c$, then $\widetilde{\text{VK}} \neq \text{VK}$ with high probability. Hence we conclude that A outputs an accepting $\tilde{c}$ such that $\widetilde{\text{VK}} \neq \text{VK}$ with probability at least $\epsilon' - \text{neg}(\kappa) \geq \epsilon'/2$. Given that $\widetilde{\text{VK}} \neq \text{VK}$, it holds that $\widetilde{\text{VK}}_i \neq \text{VK}_i$ with probability at least $\frac{\epsilon'/2}{\kappa}$ (this is because i is chosen at random). As A^* queries $\mathcal{O}_{p,\ell(\kappa)}$ only when $\widetilde{\text{VK}}_i \neq \text{VK}_i$, it holds from claim 3 that all queries of A^* to $\mathcal{O}_{p,\ell(\kappa)}$ are valid. As the oracle always correctly solves the query DL-instance, it follows that d' is a correct guess of b_i with probability at least $\epsilon'/2\kappa = \epsilon/4\kappa$. Hence, (the non-uniform machine) A^* guesses the hardcore bit of (p, g, y) with probability at least $\epsilon/4\kappa$, which is non-negligible. By the definition of the hard-core bit, it follows that the solution x to (p, g, y) can be computed with non-negligible probability. As this contradicts the GAP-DL assumption, we conclude that $\epsilon \leq \text{neg}(\kappa)$. This implies that our scheme is non-malleable with respect to commitment.

[7] For a random variable $\tilde{b}$, $D(\tilde{b})$ denotes executing the algorithm D on an input sampled according to $\tilde{b}$.

Computational hiding of our scheme is implied by the fact that it is non-malleable with respect to commitment. Hence we omit the proof. We remark that computational hiding, in fact, follows directly from (the weaker assumption) DLA. The proof goes exactly as above, except that A^* does not need the oracle $\mathcal{O}_{p,\ell(\kappa)}$, and can work directly with a distinguisher D' who contradicts hiding. ∎

String Commitment. We now extend our NINM bit-commitment scheme to obtain a string commitment scheme, that is perfectly binding, computationally hiding, and non-malleable with respect to commitment. Let $v = v_1 \circ \ldots \circ v_l$ denote the (l-bit) string to be committed. to commit to a single bit, b, κ shares of b were The string commitment scheme is the direct extension of previous scheme. To commit to v, we do the natural thing: we commit to each bit v_i, independently as in previous scheme, except that the values $\vec{p}, \vec{g}$ are chosen only once and for all v_i. Vector $\vec{y}$, of course, will be chosen independently for each v_i

The commitment scheme is described in figure 7. The committer, C, on input a string v, executes **SCom**(v) to compute and send the commitment value, c, to R. The receiver, R, on receiving the value $c = (\text{VK}, m, \sigma)$, R performs tests for each bit v_i of v exactly as in the bit-commitment scheme. If all tests succeeds, it accepts the commitment. Otherwise it outputs $\perp$, denoting "failure". Remember that each prime $p_i = 2q_i + 1$ is a safe prime.

To decommit, C sends $(\vec{x}_1, \ldots, \vec{x}_\kappa)$ to R, where $\vec{x}_i = (x_{i,1}, \ldots, x_{i,\kappa})$. R computes each bit v_i of the committed string as follows. First, it tests that $\forall i, j$ it holds that $y_{i,j} = f_i(x_{i,j})$ and c' is an l-bit string. If so, it sets v_i to be $c'_i \oplus (\oplus_{j=1}^{\kappa} \text{HCB}_{f_i}(x_{i,j}))$. If all bits are computed successfully, $v = v_1 \circ \ldots \circ v_l$. Otherwise, it outputs $\perp$, denoting failure.

Commitment: SCom(v).

Step 1: Initialize functions
Compute $(\text{VK}, \text{SK}) \leftarrow \mathcal{K}(1^\kappa)$ and $(p_1, \ldots, p_\kappa) \leftarrow \text{PRIMES}(\text{VK})$.
For all $i \in [\kappa]$, define g_i to be a random generator of $G_{q_i} \subset \mathbb{Z}^*_{p_i}$.
Let $\vec{p} = (p_1, \ldots, p_\kappa)$ and $\vec{g} = (g_1, \ldots, g_\kappa)$.
Define $f_i(x) = {g_i}^x \mod p_i$.

Step 2: Create the commitment
Let $v = v_1 \circ \ldots \circ v_l$. For each bit v_i, do the following.
Choose $x_{i,1}, \ldots, x_{i,\kappa}$ independently at random so that,

$$c'_i = v_i \oplus \left(\oplus_{j=1}^{\kappa} \text{HCB}_{f_i}(x_{i,j})\right)$$

Let $c' = c'_1 \circ \ldots \circ c'_\kappa, \vec{y}_i = (y_{i,1}, \ldots, y_{i,\kappa})$ where $y_{i,j} = f_i(x_{i,j})$.
Define $\vec{y} = (\vec{y}_1, \ldots, \vec{y}_\kappa)$ and $m = \vec{p} \circ \vec{g} \circ \vec{y} \circ c'$.
Compute $\sigma \leftarrow \text{SIGN}(m, \text{SK}, \text{VK})$.
The commitment value is, $c = (\text{VK}, m, \sigma)$. Send c to R.

Decommitment: SDcom(c).

Send $(\vec{x}_1, \ldots, \vec{x}_\kappa)$ to R, where $\vec{x}_i = (x_{i,1}, \ldots, x_{i,\kappa})$

Figure 7: Our NINM String-commitment Scheme

The following theorem establishes that (**SCom**, **SDcom**) is a NINM string commitment scheme.

Theorem 13 *If* GAP-DL *holds, the scheme* (**SCom**, **SDcom**) *is a non-interactive, perfectly binding, and computationally hiding* string *commitment scheme that is* non-malleable with respect to commitment.

Proof. The proof of this theorem is largely similar to the proof of theorem 12. The key-observation is that because only a single VK is used for *all* the bits of v, and that each bit is committed individually as before, any dependence of $\tilde{v}$ on v will need to infer some information about hard-core bits across distributed all primes. The proof is slightly different from the previous proof.

Perfect binding is straightforward. Computational hiding, as argued previously, will follow from non-malleability with respect to commitment. Observe that whenever c is an accepting commitment, it represents a well-defined and unique string v. So we directly turn to show that the scheme is non-malleable with respect to commitment.

Suppose that a man-in-the-middle A, on input an accepting commitment c, outputs $\tilde{c}$ which is an accepting commitment. As both c and $\tilde{c}$ are accepting, they correspond to well-defined and unique values, say v and $\tilde{v}$ respectively. To prove non-malleability with respect to commitment, we only need to show that (whenever $\tilde{c}$ is accepting as above) the value $\tilde{v}$ is computationally independent of value v. Recall that this is done via the following game. A produces a value v and receives a commitment c to either v or 0^l. It then produces a commitment $\tilde{c}$. Let $\tilde{v}_0$ be a random variable denoting the value committed to by A when c is a commitment to 0^l. Analogously, define, define $\tilde{v}_1$ to be a random variable denoting the value committed to by A when c is a commitment to v. We will show that $\tilde{v}_0, \tilde{v}_1$ are computationally indistinguishable.

To prove this, we first describe a (string) commitment scheme $\langle C', R' \rangle$, which is semantically secure assuming GAP-DL. We will then show that if $\tilde{v}_0, \tilde{v}_1$ are not computationally indistinguishable, then $\langle C', R' \rangle$ is not semantically secure. Scheme $\langle C', R' \rangle$ is the following.

Algorithm C', on input a string v, acts as follows. It chooses a safe prime p of size $\ell'(\kappa)$ such that GAP-DL holds in $G_q \subset \mathbb{Z}_p^*$; let g be a random generator of G_q defining the function $f \stackrel{\text{def}}{=} g^x \mod p$. For each bit v_i, it selects x_i independently at random and sets $c'_i = v_i \text{HCB}_f(x_i)$. Let $c' = c'_1 \circ \ldots \circ c'_l$, $\vec{y} = (y_1, \ldots, y_\kappa)$ where $y_i = f(x_i)$. The commitment value c'' is $(p, g, \vec{y}, c')$. To decommit, C' simply sends $(x_1, \ldots, x_\kappa)$ and R' verifies that all y_i were computed properly. If yes, it obtains v by computing each bit $v_i = c'_i \oplus \text{HCB}_f(x_i)$. It is easy to see that $\langle C', R' \rangle$ is perfectly binding and computationally hiding against adversaries A' who are PPT with oracle access to $\mathcal{O}_{p,\ell(\kappa)}$. The proof assumes that GAP-DL holds, and uses a simple hybrid argument. We omit details.

The rest of the proof for non-malleability is now largely similar to the proof of theorem 12. An adversary A breaking the non-malleability of (**SCom**, **SDcom**) can be converted into an adversary A^* who will contradict computational hiding of $\langle C', R' \rangle$ (and hence the GAP-DL assumption). A^* works by internally incorporating A, and generating a commitment for A exactly as in **SCom** except for the following difference. A^* guesses the target index i at random, and computes $x_{j,k}$ and $y_{j,k} = f_k(x_{j,k})$, for $j \in [l], k \in [\kappa], k \neq i$ exactly as in **SCom**. Define bits $b_j = \oplus_{j \neq i} \text{HCB}_{f_j}(x_{j,k})$ for $j = 1, \ldots, l$. Define string $s = b_1 \circ \ldots \circ b_l$, $v_1 = s \oplus 0^l$, and $v_2 = s \oplus v$. A^* sends v_1, v_2 to the external C' and obtains a commitment $(p, g, \vec{y}, c')$ to either v_1 or v_2. It sets $p_i = p, g_i = g, \vec{y}_i = \vec{y}, c' = c'$ to define its own commitment c completely (according to **SCom**). Note that if C' commits to v_1, c is a commitment to 0^l; and if C' commits to v_1, c is a commitment to v. A^* runs A on input c to obtain a commitment $\tilde{c}$ which is a commitment to a value distributed according to either $\tilde{v}_0$ or $\tilde{v}_1$ (depending on v_1 or v_2). By proceeding exactly as in theorem 12 (using the oracle $\mathcal{O}_{p,\ell(\kappa)}$), it is easy to see that a PPT distinguisher D for distinguishing $\tilde{v}_0$ from $\tilde{v}_1$ distinguishes commitments to v_1 from those to v_2 (according to $\langle C', R' \rangle$). This concludes the proof. ∎

6 Round Complexity of Black-box Secure Multiparty Computation

The round-complexity of secure multiparty computation (MPC) has been an active area of research.[8] Katz, Ostrovsky, and Smith [KOS03] demonstrated that secure MPC for any functionality can be performed in $O(\log n)$-rounds using *black-box* techniques (assuming standard complexity assumptions). They additionally

[8]Here, and everywhere else, we only deal with the case of MPC with abort and with no fairness. Further, we are working in the plain model, i.e., with no setup assumptions; and we always deal with the case of dishonest majority where $n-1$ out of a total n parties can be controlled by the adversary, A.

show that using *non-black-box* techniques (under super-polynomial hardness assumptions) secure MPC requires only $O(1)$-rounds. The network model assumes the availability of an (standard) ideal broadcast channel. This was further improved by Pass [Pas04] who achieved $O(1)$-round secure MPC under standard *polynomial* hardness assumptions.[9] Pass's result also uses non-black box techniques; secure MPC using only black-box techniques is not known to exist in less than $O(\log n)$-rounds.

Our simulation extractable argument π_{SE}, when used with the techniques of [KOS03], directly yields an $O(1)$-round protocol for securely computing any functionality. Further, it only uses *black-box* techniques: to simulate the view of the adversary according to the ideal/real-paradigm, the simulator S, uses the adversary A only as a black-box. We briefly mention the approach of [KOS03] and why our protocol π_{SE} applies to their construction directly.

The protocol of [KOS03] works by computing a specific functionality, called "simulatable coin-flipping", which (they show) implies secure MPC with a constant factor blow up in the number of rounds (This basically follows from [BMR90, CLOS02], and makes additional assumptions that trapdoor permutations and dense public-key cryptosystems exist.) Given a protocol for a slightly different functionality called NMCF, [KOS03] show how to achieve simulatable coin-flipping. The functionality NMCF is an n-fold parallel repetition of (slightly modified) Barak's "non-malleable coin-flipping" functionality [Bar02]: Let A be a man-in-the-middle, participating in n parallel coin-tossing protocols on "left" and n parallel coin-tossing protocols on "right"; and let $\sigma_1, \ldots, \sigma_n$ be the outputs on left, and $\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n$ be the outputs on right. Then, the functionality requires that $(\sigma_1, \ldots, \sigma_n)$ should be computationally indistinguishable from a set of n independently chosen strings; further each $\tilde{\sigma}_i$ is either a copy of some σ_j or computationally indistinguishable from a randomly and independently chosen string. In [KOS03], the NMCF functionality is a slightly modified version of Barak's protocol. But in fact, because our protocol is simulation-extractable even under *n-fold parallel repetition*, we can directly employ our protocol in the NMCF construction of [KOS03] to get a construction that uses only black-box techniques. That is, in Protocol 4 of [KOS03], the first two steps are ignored, and in the last phase (steps L5.1-5.9,R5.2-R5.10), we use our protocol π_{SE}. To see that π_{SE} is simulation-extractable under n-fold *parallel* repetition, observe that extraction of witnesses on right, can be performed on a "one-by-one" basis. Further, during simulation, because we are dealing with parallel (not *concurrent*) executions, extraction of *all* trapdoors can also be performed (either at once, or on a one-by-one basis). We thus obtain the following version of the [KOS03]-theorem (a formal proof is deferred until the full version of this paper).

Theorem 14 *For any polynomial time function f, there exists an $O(1)$-round protocol for computing f securely, tolerating $(n-1)$ (out of a total n) dishonest parties. The proof uses only* black-box *techniques.*

7 Gap-DL Resists Generic Attacks

In this section, we demonstrate that the GAP-DL assumption cannot be solved by polynomial time *generic* algorithms. A generic algorithm is one that does not make use of the specific *encoding* of group elements. Let $\mathbb{Z}_p^*$ be the multiplicative cyclic group of integers modulo a prime p; and let E be a set of bit strings of cardinality at least p. An *encoding function* of $\mathbb{Z}_p^*$ on E is an injective map ξ from $\mathbb{Z}_p^*$ into E.

Generic Group Model [Sho97]. In the generic model of groups, elements $x \in \mathbb{Z}_p^*$ are not directly given to the adversary, A. Instead they appear to A to be encoded as arbitrary *unique* strings, so that no property other than equality can be directly tested by A. The encoding may use random-looking strings, or even sequential integers. The adversary A, also known as a *generic algorithm* for $\mathbb{Z}_p^*$ on E, is a probabilistic algorithm that behaves as follows. The algorithm takes as input an *encoding list* $(\xi(x_1), \ldots, \xi(x_k))$ where each $x_i \in \mathbb{Z}_p^*$ and ξ is an encoding function of $\mathbb{Z}_p^*$ on E. As A executes, it can access an *oracle* $\mathcal{O}_{\mathbb{Z}}$, by sending it two indices i and j into the encoding list, and a sign bit specifying the operation (multiplication

[9] Pass's construction actually achieved the first *bounded concurrent* secure MPC, which was also constant round.

or division). The oracle, $\mathcal{O}_{\mathbb{Z}}$, computes $\xi(x_i \cdot x_j)$ or $\xi(x_i/x_j)$ depending upon the sign bit, and appends the resulting string to the encoding list, which A can access any time. After A finishes its execution, it outputs a bit string, denoted by $A(p, \xi(x_1), \ldots, \xi(x_k))$. We make a slight change to ξ – we think of our encoding function ξ as a pair (g, ξ'), where g is generator of $\mathbb{Z}_p^*$ and $\xi' : \mathbb{Z}_p^* \to E$ is an injective map. Now, for $a \in \mathbb{Z}_p^*$, $\xi(a) = \xi'(g^a \mod p)$.

We extend the model slightly, to take into account the access to the DL-solver oracle $\mathcal{O}_p$. Recall that no DL-queries to $\mathcal{O}_p$ can be made in the target group $\mathbb{Z}_p^*$. As DL-queries to $\mathcal{O}_p$ are generated by A, it is not necessary to encode the group elements of any of $\mathbb{Z}_q^*$ (such that $q \neq p$) in which A asks a DL-query. Thus, an element x in $\mathbb{Z}_q^*$ (such that $q \neq p$) is represented directly by its binary string representation.

The running time of A is measured by counting both – the number of bit operations and the total number of oracle queries (to both $\mathcal{O}_{\mathbb{Z}}, \mathcal{O}_p$). The following theorem establishes the unconditional hardness of (a *stronger* version of) the GAP-DL assumption in the generic model of groups. (By stronger we mean that, the size restrictions on the primes are dropped, and we are simply working with $\mathbb{Z}_p^*$ instead of its prime-order subgroup. The same proof goes through even for prime-order subgroups of $\mathbb{Z}_p^*$.

Theorem 15 *Let p be a positive prime and $E \subset \{0,1\}^*$ be a set of cardinality at least p. Let A be a generic algorithm for $\mathbb{Z}_p^*$ on E making at most m queries to $\mathcal{O}_p$ and $\mathcal{O}_{\mathbb{Z}}$, all counted together. If $x \in \mathbb{Z}_p^*$ and the encoding function ξ are chosen at random, then the probability that $A(\xi(1), \xi(x)) = x$ is $O(m^2/p)$. The probability is taken over the random choices of x, ξ, as well as the randomness of A.*

Proof. To prove theorem, we proceed along the lines of Shoup [Sho97]. Instead of giving A access to $\mathcal{O}_p$, consider an algorithm B that plays the following game with A. Let X be an indeterminant. B maintains two lists F and L. At any step in the game, F contains $F_1, \ldots, F_k$ such that each F_i is a linear polynomial in $\mathbb{Z}_p^*[X]$, and L contains ξ_1, ξ_k of *distinct* values in E that are given out to the adversary. To start the game, B sets $k = 2, F_1 = 1, F_2 = X$ and chooses ξ_1, ξ_2 at random such that $\xi_1 \neq \xi_2$. B now provides ξ_1, ξ_2 to A. A now performs its computation and may make queries to either $\mathcal{O}_p$ or $\mathcal{O}_{\mathbb{Z}}$ (which is simulated by B). Every query of A to $\mathcal{O}_p$, is directly answered by $\mathcal{O}_p$ who computes the discrete logarithm, provided A respects the query-structure set-forth in section 3. Every query (i, j, b) of A, intended for $\mathcal{O}_{\mathbb{Z}}$, is instead answered by B as follows. B computes F_{k+1} as either $F_i + F_j$ or $F_i - F_j$ (according to b). If $F_{k+1} = F_l$ for $1 \leq l \leq k$, B sets $\xi_{k+1} = \xi_l$; otherwise, it sets ξ_{k+1} to a random element in E, distinct from $\xi_1, \ldots, \xi_k$. String ξ_{k+1} is then given to A.

When A finishes its execution, it outputs a string $y \in \mathbb{Z}_p^*$. B now does the following. It chooses $x'\mathbb{Z}_p^*$ at random and tests whether $F_i(x) = F_j(x)$ for any $F_i \neq F_j$ or if $x' = y$. If the test succeeds, we say that A wins the game and loses otherwise.

Observe that if for some i, j it holds that $F_i(x) = F_j(x)$ and $F_i \neq F_j$, the simulation of $\mathcal{O}_p$ by B is flawed since it provides to A two different representations of the same element. As for any fixed i, j such that $F_i \neq F_j$, the polynomial $F = F_i - F_j$ is of degree 1, the probability that $F(x') = 0$ is at most $1/p$. Similarly, $x' = y$ with probability exactly $1/p$. Now observe that if the simulation is not flawed, A correctly solves if and only if it wins the game, which happens with $1/p$ probability. Further the simulation is flawed with probability at most m^2/p counting over all i, j. Hence, the probability of winning the game is $O(m^2/p)$, which bounds from above the probability that $A(p, \xi(1), \xi(x)) = x$. ∎

Acknowledgements

I am indebted to Vipul Goyal for several useful discussions and comments. In particular, section 6 of this paper is due to a suggestion of his. I would also like to thank Ryan Moriarty for discussions regarding [MMY06], and Darrel Carbajal for mentioning [OP01a] to me.

References

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of dhies. In *CT-RSA*, pages 143–158, 2001.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *Proc. 42nd FOCS*, pages 106–115, 2001.

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *Proc. 43rd FOCS*, 2002.

[BGGL01] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-sound zero-knowledge and its applications. In *Proc. 42nd FOCS*, pages 116–125, 2001. Preliminary full version available as Cryptology Eprint Archive Report 2001/063.

[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.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. Cryptology ePrint Archive report., 2006. `http://eprint.iacr.org/`.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[BSZ07] Joonsang Baek, Ron Steinfeld, and Yuliang Zheng. Formal proofs for the security of signcryption. *J. Cryptology*, 20(2):203–235, 2007.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Bob Werner, editor, *FOCS*, pages 136–147, 2001. Preliminary full version available as Cryptology ePrint Archive Report 2000/067.

[CKPR03] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. *SIAM Journal on Computing*, 32(1):1–47, February 2003. Preliminary version in STOC '01.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party computation. In *Proc. 34th STOC*, pages 494–503, 2002.

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

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

[DIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *STOC*, pages 141–150, 1998.

[DKOS01] Giovanni Di Crescenzo, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Efficient and non-interactive non-malleable commitment. Report 2001/032, Cryptology ePrint Archive, April 2001. Preliminary versoin in EUROCRYPT 2001.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. pages 409–418, 1998.

[FF00] Marc Fischlin and Roger Fischlin. Efficient non-malleable commitment schemes. *Lecture Notes in Computer Science*, 1880:413–430, 2000. Extended abstract in CRYPTO' 2000.

[FLS99] Feige, Lapidot, and Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29, 1999.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.

[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.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. Earlier version available on `http://www.wisdom.weizmann.ac.il/~oded/frag.html` .

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

[JJ02] Jakob Jonsson and Burton S. Kaliski Jr. On the security of rsa encryption in tls. In *CRYPTO*, pages 127–142, 2002.

[KOS03] J. Katz, R. Ostrovsky, and A. Smith. Round efficiency of multi-party computation with a dishonest majority. In *Eurocrypt '03*, 2003.

[KP05] Caroline Kudla and Kenneth G. Paterson. Modular security proofs for key agreement protocols. In *ASIACRYPT*, pages 549–565, 2005.

[MMY06] Tal Malkin, Ryan Moriarty, and Nikolai Yakovenko. Generalized environmental security from number theoretic assumptions. In *TCC '05*, 2006.

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

[MY08] Daniele Micciancio and Scott Yilek. The round-complexity of black-box zero-knowledge: A combinatorial characterization. In *TCC*, pages 535–552, 2008.

[OP01a] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *Public Key Cryptography*, pages 104–118, 2001.

[OP01b] Tatsuaki Okamoto and David Pointcheval. React: Rapid enhanced-security asymmetric cryptosystem transform. In *CT-RSA*, pages 159–175, 2001.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.

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

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *Proc. 37th STOC*, 2005.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.

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

[SBZ02] Ron Steinfeld, Joonsang Baek, and Yuliang Zheng. On the necessity of strong assumptions for the security of a class of asymmetric encryption schemes. In *ACISP*, pages 241–256, 2002.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[SWP04] Ron Steinfeld, Huaxiong Wang, and Josef Pieprzyk. Efficient extension of standard schnorr/rsa signatures into universal designated-verifier signatures. In *Public Key Cryptography*, pages 86–100, 2004.

A On Simulators S_2 and S_3

A Technical Modification. We discuss one (purely technical) change in the description of both S and S_1. Both the simulators internally emulate the actions of an honest V for the right hand side session. During the protocol execution, V generates a statement-witness pair $(\overline{\text{DL}}, \overline{a})$ for relation $R_{\overline{\text{DL}}}$, and later uses $\overline{a}$ to prove an intermediate statement $(\overline{\text{DL}}, \overline{c}_1, \overline{c}_2)$ using the 3-round BH-protocol (see description of π_{SZA}). To aid ourselves, instead of viewing this as an internal component of V, we will view this as coming from an oracle, $\mathcal{O}_V$. Formally, the part of V that computes $(\overline{\text{DL}}, \overline{a})$, sends a query (st_1) to $\mathcal{O}_V$, where st_1 is the (current) state of the simulator. The oracle $\mathcal{O}_V$ computes the DL-instance exactly as V would have, and sends (only) $\overline{\text{DL}}$ to the simulator (and hence V) who uses $\overline{\text{DL}}$ to continue the simulation. Later, when V needs a witness $\overline{a}$ to complete the proof for $(\overline{\text{DL}}, \overline{c}_1, \overline{c}_2)$, it sends another query (st_2) to $\mathcal{O}_V$ who then responds with a witness $\overline{a}$ which V can use. Once again, st_2 denotes the current state (or global view) of the simulator (which includes internal adversary A's view). Note that this does not affect any of our previous claims. Hence, from now on, $S \stackrel{\text{def}}{=} S^{K_{\text{SZA}}, \mathcal{O}_V}$ and $S_1 \stackrel{\text{def}}{=} S_1^{\mathcal{O}_{\tilde{p},\ell(\kappa)}, \mathcal{O}_V}$. Now we present the formal descriptions of S_2, S_3. The simulator S_2 is identical to S_1 except that instead of having access to $\mathcal{O}_V$, it has access to a slightly different oracle $\mathcal{O}'_V$. Similarly, S_3 and S_2 are identical except that S_3 has access to a slightly different oracle $\mathcal{O}''_V$, instead of $\mathcal{O}'_V$.

Simulator $S_2^{\mathcal{O}_{\tilde{p},\ell(\kappa)}, \mathcal{O}'_V}$. For notational ease, let $S_2 \stackrel{\text{def}}{=} S_2^{\mathcal{O}_{\tilde{p},\ell(\kappa)}, \mathcal{O}'_V}$. Simulator S_2 is *identical* to S_1 except that it has access to $\mathcal{O}'_V$ instead of $\mathcal{O}_V$. Oracle $\mathcal{O}'_V$ behaves as follows:

1. For the first query st_1, it behaves identically to $\mathcal{O}_V$ and returns $\overline{\text{DL}}$.
2. For the second query, st_2, $\mathcal{O}'_V$ uses the `EXTRACT` routine of the 2-slot simulator of π_{SZA} using the state information st_2 and the oracle $\mathcal{O}_{\tilde{p},\ell(\kappa)}$ to compute a witness $w_2 = (\sigma, \omega, i, j)$ for the statement $\overline{\text{DL}}, \overline{c}_1, \overline{c}_2)$. If $w_2 = \perp$, $\mathcal{O}'_V$ answers with $\perp$. Otherwise, it answers with the original witness $\overline{a}$.

Simulator $S_3^{\mathcal{O}_{\tilde{p},\ell(\kappa)}, \mathcal{O}''_V}$. For notational ease, let $S_3 \stackrel{\text{def}}{=} S_3^{\mathcal{O}_{\tilde{p},\ell(\kappa)}, \mathcal{O}'_V}$. Simulator S_3 is *identical* to S_2 except that it has access to $\mathcal{O}''_V$ instead of $\mathcal{O}'_V$. Oracle $\mathcal{O}''_V$ is the same as $\mathcal{O}'_V$ except that when answering the second query st_2, if the witness $w_2 \neq \perp$, it responds with w_2 (and not $\overline{a}$).