Fair-Zero Knowledge

Matt Lepinski[1], Silvio Micali, and abhi shelat

Massachusetts Institute of Technology, Cambridge MA 02114, USA,
{lepinski,silvio,abhi}@csail.mit.edu,
Home page: http://crypto.csail.mit.edu/~abhi

Abstract. We introduce *Fair Zero-Knowledge*, a multi-verifier ZK system where every proof is guaranteed to be "zero-knowledge for all verifiers." That is, if an honest verifier accepts a fair zero-knowledge proof, then he is assured that all other verifiers also learn nothing more than the verity of the statement in question, even if they maliciously collude with a cheating prover.

We construct Fair Zero-Knowledge systems based on standard complexity assumptions (specifically, the quadratic residuosity assumption) and an initial, one-time use of a physically secure communication channel (specifically, each verifier sends the prover a private message in an envelope). *A*ll other communication occurs (and must occur) on a broadcast channel.

The main technical challenge of our construction consists of provably removing any possibility of using *steganography* in a ZK proof. To overcome this technical difficulty, we introduce tools —such as Unique Zero Knowledge— that may be of independent interest.

1 Introduction

A New Worry A traditional zero-knowledge proof enjoys two crucial properties, *soundness* and *zero knowledge*, each guarding the interests of mutually cautious parties. Soundness protects the verifier: a malicious prover has practically no chance to convince the verifier of a false statement. Zero knowledge protects the prover: a malicious verifier has practically no chance of learning anything about the statement in question beyond the fact that it is indeed true.

A new threat emerges, however, when there are *multiple* verifiers. In such a situation, a malicious prover may *collude* with some of the verifiers by generating proofs that convey additional information to them while remaining zero-knowledge to all others. Indeed, an honest verifier that accepts a ZK proof of a given theorem learns nothing more that the verity of the theorem statement in question, but *can he be sure that the same holds for his "colleagues?"*

Notice that the traditional definition of a zero-knowledge proof is *orthogonal* to the above concern. Let us illustrate this point by constructing the following (somewhat artificial) NIZK proof system, (P', V') —which uses as a subroutine (P, V), the original NIZK proof system of [BSMP91].

P' initially chooses (PK, SK), the public and secret key of a uniquely decryptable public-key cryptosystem. Later on, whenever it receives as an input a member x of a NP-language L together with a witness w for $x \in L$, P' first computes w', an encryption of w relative to PK, and then outputs a proof string π' which consists of (1) PK, (2) w', and (3) a NIZK proof — according to (P, V) and some common reference string σ— of the statement "there is a decryption key corresponding to PK (i.e., SK) such that after decrypting w' with said key, one obtains a witness for $x \in L$."

Clearly each such π' is accepted by (and is zero-knowledge for) all honest verifiers. But, a malicious prover P' may, without notice, ensure that it is "much more informative" for some colluding verifiers. There are very subtle ways for him to accomplish this, but the simplest one consists of having P' provide each colluding verifier with SK, so that each subsequent π' reveals the corresponding witness w in its entirety to each colluding verifier!

A New Goal We wish to build a multi-verifier zero-knowledge system that is provably *fair*. That is, we wish to guarantee that whenever an honest verifier accepts a fair ZK proof, then he is assured that all other verifiers too (whether honest or colluding with a malicious prover) learn nothing more than the verity of the statement in question. In other words, we wish to *extend the zero-knowledgeness property of a ZK system to protect also the Verifier(s) and not just the Prover!*

A Motivating Example In repeated auctions of similar items,[1] it may be desirable that all bids in an individual auction (including the winning bid) remain secret in subsequent ones. This goal appears to be a golden opportunity for encryption and zero-knowledge proofs, but special care must be taken. The following example illustrates.

A closely-watched auctioneer possessing a public encryption key PK sells a series of n lithographs from the same etching by repeating the following two-step process, once for each lithograph. First, the bidders publicly announce their individual bids encrypted with PK; second, the auctioneer proves in zero knowledge who the winner of the current lithograph is. At the very end, the auctioneer privately collects all right amounts from the winners.

Using zero-knowledge proofs in Step 2 aims at providing the minimum amount of knowledge enabling the bidders to decide whether they should continue bidding in subsequent auctions. At a superficial level, this aim seems to be achieved: First, if the auctioneer is honest, then *standard* zero-knowledge proofs guarantee that no additional bid information is leaked prematurely. Second, even if the auctioneer were dishonest, by virtue of being closely-watched he could not use any "side channels" to divulge additional bid information to a selected subset of the

[1] A well-studied problem in economics

bidders. A better analysis, however, shows that our new worry naturally arises in this setting: no matter how closely watched, a dishonest auctioneer might use the ZK proof *itself* as a mechanism to leak bid information to a colluding bidder (thus giving him an advantage in later rounds).

In sum, a *standard* ZK proof of who is winner of an individual auction is not enough here: what is really needed is a *Fair* ZK proof!

Fair Zero Knowledge Syntactically, Fair ZK is a two-phase process. The first phase consists of a *preprocessing* protocol, where various quantities (e.g., public keys) are established, and private channels are (seemingly necessarily) used. The second phase consists of a *proving* protocol, where it is imperative that the Prover be restricted to communicate via broadcast only. (Were a malicious prover connected to some colluding verifiers via private channels during the proving phase, it would be impossible to prevent the selective dissemination of witness information!) In a sense, Prover and Verifiers might execute the preprocessing phase on Earth, but for the proving phase, Prover is sent to the moon from where anything he says is heard by all Verifiers.

Semantically, Fair Zero Knowledge guarantees that, for any NP-theorem that (1) has a single witness and (2) is chosen after preprocessing ends, the prover cannot undetectably communicate anything more than the truthfulness of the theorem in question to any verifier, no matter what arrangements they might have made beforehand.

Postponing for a moment a discussion of our "unique-witness constraint," notice that Fair Zero Knowledge does not provide any guarantees for theorems whose witnesses are known beforehand to the prover. In this case, the (to be) prover could have already divulged witness information to anyone he wanted, and protocol designers have no responsibility for what happens before the protocol starts! [2]

A bit more precisely, assume that, for a unique-witness NP-language L and for all $i = 1, 2, \ldots$, (a) $x_i \in L$ is chosen on-line and its statement made public, and (b) w_i, the witness of x_i, is privately given to the Prover. Then, Fair ZK enables the Prover to prove to a set of verifiers that every x_i, *individually* and *in order*, indeed belongs to L, so that

1. *When the Prover is honest, no set of malicious verifiers can learn anything more than the mere sequence of statements "$x_i \in L$ and it is the ith theorem;" and*
2. *No dishonest prover can convey any knowledge other than $x_i \in L$ to any colluding verifier —no matter how much information they secretly share up*

[2] In this extended abstract we wish to keep the notion of Fair ZK as simple as possible. In the final version, however, we shall allow colluding verifiers to have prior partial information y about a proven theorem x, and extend Fair ZK to ensure that they cannot get any knowledge from the proofs beyond y and x's truthfulness. The difficulty of doing this right lies in the fact that a malicious prover himself may know some other information z about x beforehand, including what information y some verifiers know.

to the last round of preprocessing— without being detected by all honest verifiers.

We are actually able to construct a non-interactive version of Fair ZK under physical-channel and standard complexity assumptions. Namely, under the quadratic residuosity assumption, there exists a Fair ZK proof system whose prover —after a preprocessing protocol in which he receives an envelope from each verifier— proves every theorem x_i non-interactively by broadcasting a proof string π_i.

Technical Challenges There are two obstacles in constructing Fair ZK.

The first, and main obstacle is *preventing steganography.* The prover of any zero knowledge proof must be probabilistic, and this very probabilism ushers in the ability for a malicious prover to communicate with colluding verifiers over "subliminal channels." That is, a malicious prover can use the probabilism of a ZK proof to convey steganographically, to a subset of verifiers, witness information about any NP theorem —even one chosen after he has been "sent to the moon." For example, in the above repeated-auction process, all theorems are of the type "the winner of the current auction is bidder X", and thus are generated after the prover (i.e., the auctioneer) starts being closely watched. Yet, a malicious auctioneer may agree beforehand with a subset of colluding bidders on a way to encode the actual amount of the highest bid in the bits he broadcasts in the corresponding ZK proof.[3] In fact, as early as 1987, Desmedt, Goutier and Bengio [DGB87] illuminated this problem by showing that a particular zero-knowledge protocol due to Fiat and Shamir can easily be used as a perfect subliminal channel. More generally, Langford, Hopper, and von Ahn [NHvA02] show that whenever there is entropy, steganographic communication —provably undetectable by honest parties— always exists.

Perhaps surprisingly, in light of their result, we show how to *provably prevent steganography in our context.* In our approach, we construct a novel type of ZK system, uniZK, in which the prover's probabilism is confined to a preprocessing phase after which not only is he made totally deterministic, but *his determinism is actually made universally verifiable.* Very roughly, in a uniZK system the prover first establishes a suitable public key, so that, for any NP-theorem x having a single witness, it is universally verifiable that there exists a single ZK way to prove x that is acceptable by an honest verifier. Such "unique provability" therefore *provably bans steganography from* uniZK *proofs.*

Note that in our application we need *verifiable determinism*, and not just determinism. Naively, one might consider constructing a uniZK system by replacing the probabilistic prover of any NIZK system with one who chooses a short random seed for a pseudo-random function [GGM86] and acts deterministically ever after. However, while this would be conceptually simple to do, it would also be impossible for an efficient verifier to *check* that the prover indeed behaved in such a fashion instead of flipping new coins for each proof. Thus, an honest

[3] For example, a naive approach is to make the first 20 bits of the proof the same as those of the winning bid.

verifier may not be convinced that a malicious prover is not steganographically conveying additional witness information to colluding verifiers. In sum, prover determinism might be easy, but verifiable prover determinism is not!

After so overcoming steganography, a second obstacle remains in building Fair Zero Knowledge. While, in a uniZK system, knowledge of the prover's public key ensures that there is only one acceptable proof, knowledge of the corresponding secret key may enable anyone to read off the entire witness from such a proof! Thus, we must ensure that the prover is the only one possessing knowledge of his secret key. If the generation of his public-secret key pair were totally up to him, however, this would be impossible, because the prover and his accomplices may agree on which public-secret keys he will choose. Instead, we show how to generate and distribute the prover's keys by a protocol involving all verifiers so that, as long as there is one honest verifier, then only the prover will know the resulting secret key. It is in this subprotocol that we make a single use of a physically secure communication channel: namely every verifier sends to the prover a single message in *an envelope*. After this, all communication (in the prover-key-generation subprotocol and in all subsequent uniZK proofs) is via broadcast.

More on Preprocessing Our protocol and even our definition of Fair ZK includes preprocessing. The reason for this is that Zero Knowledge, as any secure protocol, requires randomness and (as discussed above) any amount of entropy enables undetectable steganography, which defeats fairness. We prove, however, that we can confine the necessary entropy "somewhere" where it is actually "innocuous". Such a place is our preprocessing phase. Though steganography may be rampant during preprocessing, it is also useless since the theorems to be proved in zero-knowledge have not yet been selected – and thus no information about their proofs can be conveyed. Nor can such information be conveyed afterward preprocessing, since all communication after preprocessing is via broadcast and verifiably unique!

More On Envelopes Usage of a physically secure channel is crucial to our preprocessing. In our application, it is unclear how to simulate these channels by an "encrypt-and-broadcast" process, since such methodology must start with the prover choosing a suitable encryption key, and he could always choose a key whose corresponding secret key is already known to his accomplices. In such a case, any message sent encrypted to the prover by an honest verifier will be understood by a dishonest one, defeating the very reason for encrypting it. By delivering a message to the prover in an envelope, however, honest verifiers are guaranteed that the message will indeed remain secret to any malicious verifier, thus "dividing the state of knowledge of the prover from that of the verifier" at a specific moment of the protocol. (The protocol must then ensure —e.g., via "steganography-free broadcasting"— that these divided states of knowledge will indeed continue to remain so!)

But if physically secure channels must be used, why envelopes rather than traditional private channels? The point is that traditional private channels are "bidirectional." We instead need to prevent a malicious prover, after receiving a private message M from an honest verifier along a physically secure channel, from forwarding M to a colluding verifier along another, similar channel. Thus, we require mono-directional channels from the verifiers to the prover. Envelopes in our protocol are just good (and well known!) examples of mono-directional physically secure channels. Let us remark that, since envelopes may be more inconvenient than broadcasting in many a setting, it is a feature of our protocol that envelope communication is confined to a single round!

More on Witness Uniqueness Let us now explain why we define Fair ZK for NP-languages whose members have a unique witness. We allow a Fair ZK proof to depend (via an underlying uniZK proof) on the given input witness w. Thus, if the prover knows two or more witnesses for $x \in L$, he can have "a multiplicity of Fair ZK proofs to choose from," which would again enable steganographic communication. For instance, the NP-complete language of 3-colorability appears to be unsuitable for Fair ZK proofs as we define them, because from any coloring of a graph one can immediately compute 5 more colorings by just permuting the three colors!

Note, however, that our unique NP-witness requirement is often automatically satisfied in cryptographic applications. This is so because underlying complexity problems (e.g., integer factorization, discrete logarithm, etc.) often have unique solutions, and appropriate NP reductions can be used so as to "preserve such uniqueness." For example, the desired ZK proofs of our motivating example are for unique-witness languages, because all bids are encrypted by means of a uniquely decryptable cryptosystems.

More generally, Fair ZK actually applies to *computationally unique-witness languages*, that is, to languages for which it is hard for the prover to generate a second witness from a first one. (For example, this encompasses statements which refer to most computationally binding commitment schemes.) In sum therefore, this enlarged constraint is very mild in a cryptographic setting, making Fair ZK widely applicable.

Notice, that while Fair ZK is quite meaningful when applied to NP-languages having computationally unique witnesses, uniZK can be meaningfully defined for all NP-languages.[4] Thus, in the next section we define uniZK for all NP-languages, and then, in Section 3, we define Fair ZK only for those languages having computationally unique witnesses.

2 Notation

We shall follow, verbatim, [BSMP91] and [GMR88]. A function $\mu(\cdot)$ from non-negative integers to reals is called *negligible* if for every constant $c > 0$ and all sufficiently large n, $\mu(n) < n^{-c}$. An *efficient* algorithm is a probabilistic algorithm running in expected polynomial time. If S is a probability space,

[4] Essentially, as we shall see, " any witness efficiently maps to a uniZK proof, and vice versa."

then "$x \leftarrow S$" denotes the probabilistic algorithm consisting of choosing an element x at random according to S and returning x. If p is a predicate, then the notation "$x \leftarrow S|p(x)$" denotes the assignment consisting of choosing an element x at random according to S, and returning the first x such that $p(x)$ is true. Let $S_1, S_2, \ldots$ be probability spaces, then the notation $\Pr[x_1 \leftarrow S_1;\ x_2 \leftarrow S_2;\ \ldots\ :\ p(x_1, x_2, \ldots)]$ denotes the probability that the predicate $p(x_1, x_2, \ldots)$ is true after the ordered execution of the assignments $x_1 \leftarrow S_1;\ x_2 \leftarrow S_1;\ \ldots$ If S, T, $\ldots$ are probability spaces, the notation $\{x \leftarrow S; y \leftarrow T; \cdots : (x, y, \cdots)\}$ denotes the new probability space over $\{(x, y, \cdots)\}$ generated by the ordered execution of the assignments $x \leftarrow S$, $y \leftarrow T, \cdots$.

3 Unique Non-interactive Zero-Knowledge

We define Unique Non-interactive Zero-Knowledge (uniZK) proofs as special types of NIZK proofs. Thus, we begin this section with a review of NIZK, and then proceed to give a precise formalization and construction of uniZK.

NIZK, in a Nut Shell An NIZK proof system [BFM88] [BSMP91] for a NP-language L consists of a pair of efficient algorithms, a prover P and a verifier V, and a public, random string, σ, called the *reference string*. When proving that the statement "x is a member of L", it is assumed that P is also privately given a witness w for $x \in L$. The proof process is extremely simple: P computes a single string π (for proof), $\pi = P(x, w, \sigma)$, and sends it to V. The verifier, on inputs x, σ and proof string π accepts or rejects, without having to reply to P (hence, non-interactively). This process can be repeated, with the same reference string, for an unbounded number of theorems (i.e., members of L).

Semantically, an NIZK satisfies the usual ZK properties of *Completeness*, *Soundness* and *Zero Knowledgeness*. In this non-interactive setting, completeness means that, for every reference string and every genuine member of L, the verifier accepts all honestly generated proofs. soundness means that, for most reference strings, no acceptable "proof" π^* exists for any $x^* \notin L$. Zero-Knowledgeness means that there exists an efficient simulator S that first generates a reference string σ' and then, for any sequence of theorems, $x_i, x_2, \ldots$, (and without any witness information) generates strings $\pi'_1, \pi'_2, \ldots$, such that the sequence $\sigma', \pi'_1, \pi'_2, \ldots$ is indistinguishable from the sequence consisting of a random reference string followed by the proofs that an honest prover would generate for the same theorem sequence —with the proper witness information!

The construction of [BSMP91] actually satisfies (but does not claim) a stronger notion of Zero Knowledgeness, that was put forward in [FLS90]. Namely, the simulator S (rather than being given the sequence of theorems $x_1, x_2, \ldots$ up-front) must produce each string π_i knowing theorem x_i but not future ones. (Thus, although we adopt this stronger notion of zero knowledgeness for uniZK, we can base our uniZK construction on the NIZK system of [BSMP91].)

Adding Verifiably Unique Provability to NIZK As anticipated in the Introduction, we wish to define uniZK for all NP-languages (rather than for those having computationally unique witnesses). We do so by demanding that, for any $x \in L$, any prover —honest or malicious— "may produce a single uniZK proof for every witness he knows." How can this be formalized?

The easiest way would be demanding that, every $x \in L$, no matter how many witnesses it may have, has a single uniZK proof. Unfortunately, no such uniZK system may exist. (We certainly do not know how to construct one.)

A second way might be demanding the existence of a unique uniZK proof for each NP-witness. Unfortunately, relative to our steganography-free goals, such a definition may not be sufficiently meaningful, because it leaves open the possibility for a malicious prover to choose from a multiplicity of uniZK proofs by "rewriting" then. Assume that an efficient, malicious prover P' were given a witness w of a theorem x belonging to an NP-language L with computationally unique witnesses. Then, w would be the only witness of $x \in L$ known to P', and by Completeness, P' could certainly produce one uniZK proof, π_w. But now, if from π_w one could also compute additional uniZK proofs for $x \in L$, P' could compute a multiplicity of uniZK proofs for $x \in L$ from a single witness!

We thus formalize uniZK by demanding that (for most reference strings σ and public keys PK) the honest algorithm P forms an *easy-to-invert bijection* between the witness set of $x \in L$ (denoted W_x) and the set of acceptable uniZK proofs (denoted $\Pi_{PK}(x, \sigma)$). This captures the notion that any prover "can only produce a single uniZK proof for any witness he knows:" his ability to produce multiple uniZK proofs from a single witness can solely originate from his ability of producing multiple witnesses from a single one.

To complete our formalization, we must handle the case of a cheating prover who posts an invalid public key PK^*; that is, a key that does not pass a proper inspection of a honest verifier. In this case, it is reasonable for the verifier to reject any subsequent proof: after all, he knows for certain that the prover is malicious! Therefore, our definition requires that either the set of acceptable proofs $\Pi_{PK^*}(x, \sigma)$ is empty, or else there exists a secret key SK^* such that $P(x, \cdot, \sigma, SK^*)$ forms an efficient bijection from W_x to $\Pi_{PK^*}(x, \sigma)$. For this to be meaningful, however, such SK^* should be unique, that is, there must be a function sk (possibly hard to compute) mapping any "reasonable looking" public key PK^* to the right SK^*.

In sum, our definition states that unless $\Pi_{PK^*}(x, \sigma)$ is empty, $P(x, \cdot, \sigma, sk(PK^*))$ forms an efficient bijection from W_x to $\Pi_{PK^*}(x, \sigma)$.

3.1 Formal Definition

Let L be an NP language, and R_L be its corresponding, polynomial-time relation. We say that a sequence of pairs of strings, $(x_1, w_1), (x_2, w_2), \ldots$, is a *theorem-witness sequence for* L if each $x_i \in L$ and $w_i \in R_L(x_i)$.

Definition 1. *A triple of efficient algorithms, (G, P, V), where P is deterministic, is a* unique non-interactive zero-knowledge (uniZK) proof system *for an NP-language L if there exists a positive constant c and a negligible function μ such that the following properties are satisfied:*

Completeness: $\forall$ *theorem-witness sequences* $(x_1,w_1),(x_2,w_2),\ldots$ *for* L*, and for all* $k>2$

$$\Pr\left[\begin{array}{l}(PK,SK)\leftarrow G(1^k);\ \sigma\leftarrow\{0,1\}^{k^c};\ \pi_1=P(x_1,w_1,\sigma,SK,1);\\ \pi_2=P(x_2,w_2,\sigma,SK,2)\ldots:\bigwedge_i V(x_i,\sigma,PK,\pi_i,i)=1\end{array}\right]=1$$

Soundness: $\forall k>2$ *and* $\forall$ *algorithms* P^*

$$Pr\left[\sigma\leftarrow\{0,1\}^{k^c};(x^*,PK^*,\pi^*,i)\leftarrow P^*(\sigma)\ :x^*\notin L\wedge V(\sigma,x^*,PK^*,\pi^*,i)=1\right]<\mu(k)$$

Zero-Knowledgeness: $\exists$ *an efficient algorithm* S *such that* $\forall$ *theorem-witness sequences* $(x_1,w_1),(x_2,w_2),\ldots$ *for* L*, the following two ensembles are computationally indistinguishable:*

$$\left\{\begin{array}{l}(PK,SK)\leftarrow G(1^k);\sigma\leftarrow\{0,1\}^{k^c};\pi_1=P(x_1,w_1,\sigma,SK,1);\\ \pi_2=P(x_2,w_2,\sigma,SK,2)\ldots\ :\ (\sigma,PK,\pi_1,\pi_2,\ldots)\end{array}\right\}_k$$

$$\left\{\begin{array}{l}(PK',SK',\sigma')\leftarrow S(1^k);\ \pi_1'\leftarrow S(SK',x_1,1);\\ \pi_2'\leftarrow S(SK',x_2,2),\ldots:(\sigma',PK',\pi_1',\pi_2',\ldots)\end{array}\right\}_k$$

Uniqueness: $\exists$ *a deterministic function* $sk(\cdot)$ *and an efficient deterministic algorithm* P^{-1} *such that* $\forall x\in L$*,* $\forall i>0$*, and* $\forall PK^*\in\{0,1\}^*$*,*

$$\Pr\left[\begin{array}{ll}\sigma\leftarrow\{0,1\}^{k^c}; & \left(|\Pi^i_{PK^*}(x,\sigma)|>0\right)\Rightarrow\\ & P(\sigma,x,\cdot,sk(PK^*),i):W_x\xrightarrow{1-1}\Pi^i_{PK^*}(x,\sigma)\ \bigwedge\\ & P^{-1}(\sigma,x,\cdot,sk(PK^*),i):\Pi^i_{PK^*}(x,\sigma)\xrightarrow{1-1}W_x\end{array}\right]>1-\mu(k)$$

where $W_x=\{w:w\in R_L(x)\}$ *and* $\Pi^i_{PK^*}(x,\sigma)=\{\pi:V(x,\sigma,PK',\pi,i)=1\}$.

3.2 Constructing uniZK

We can construct a uniZK system based on the hardness of the quadratic residuosity problem[GM84], for Blum integers, by modifying the protocol of Blum, De Santis, Micali and Persiano [BSMP91]. [5]

Theorem 1. *If quadratic residuosity is hard, then there exist* uniZK *systems for 3SAT.*

Proof Sketch: The key generator, $G(1^k)$, produces a public key consisting of a randomly selected k-bit Blum integer, x, and a quadratic non-residue, y mod x. We denote the tuple (x,y) as a *proving pair.* The secret key consists of the factorization of x.

Let $(a_1,\ldots,a_m)$ be a tuple of k-bit integers that have Jacobi symbol 1 mod x. If $(b_1,\ldots,b_m)$ is tuple of bits then we say that $(a_1,\ldots,a_m)$ *has type*

[5] We can also make a uniZK system for CIRCUIT-SAT by combining the single-theorem protocol of Damgärd [DAM92] with the multi-theorem techniques of Blum, De Santis, Micali and Persiano.

$(b_1, \ldots, b_m)$ if each a_i is a square mod x if and only if b_i is 0. If $(c_1, \ldots, c_m)$ is a tuple of k-bit integers then we say that $(a_1, \ldots, a_m)$ and $(c_1, \ldots, c_m)$ *have the same type* if a_i is a square mod x if and only if c_i is a square mod x.

A prover who knows the factorization of x can prove that the tuple $(a_1, \ldots, a_m)$ has type $(b_1, \ldots, b_m)$ by providing, for each i, a square root of $a_i y^{b_i}$ mod x. Similarly, a prover can prove that $(a_1, \ldots, a_m)$ and $(c_1, \ldots, c_m)$ have the same type by providing, for each i, a square root of $a_i c_i \mod x$. To make these proofs unique, whenever the prover provides a square root, he provides the Jacobi-symbol 1 square root which is less than $n/2$. (Since x is a Blum integer, there is exactly one such root for every quadratic residue.) The verifier rejects any proof in which a different square root is provided.

Following [BSMP91], we first present a proving algorithm, P, for the single theorem case. Let 3-SAT be the language of satisfiable boolean 3-CNF formulas. Let $\phi \in 3 - SAT$ be a theorem with m clauses and variables $v_1, \ldots, v_n$ and let w be a satisfying assignment for ϕ.

1. Break the reference string into two parts, ρ and τ where $|\rho| = 16k^3$ and $|\tau| = 64k^2n + 48k^3m$.
2. Parse ρ into k-bit integers; skip any values that are greater than x or have Jacobi symbol -1.
3. Prove that each of the remaining k-bit integers in ρ has either type 0 or type 1 by giving a square root mod x or a square root of it times y mod x. As in [BSMP91], this proves that (x, y) is a properly-formed *proving pair*, that is, that x is a Blum Integer and y is a quadratic non-residue mod x.
4. Parse τ into k-bit integers as in Step 2.
5. Acquire n pairs of k-bit integers such that each pair is either of type $(1, 0)$ or type $(0, 1)$. To do this, parse a section of τ as $8kn$ pairs. Then for each pair (s, t) (in order) either give a square root of st mod x and discard the pair or give a square root of sty mod x and select the pair. Once n pairs have been selected, discard any remaining pairs.
6. Now define a value u_i corresponding to each variable v_i in ϕ as follows: let u_i be the quadratic residue in the ith pair acquired in Step 5 if v_i is false in w, and to the non-residue in the pair otherwise.
7. Let v_d, v_e and v_f be the three variables that appear in clause j of ϕ. For each clause j of ϕ, form a triple (a_j, b_j, c_j) where a_j is equal to u_d if v_d appears non-negated in the clause or to the product of u_d and $y \mod x$ otherwise. The values b_j and c_j are analogously defined.
8. Parse the remaining portion of τ as $8k^2m$ triples of k-bit integers. Among the jth set of $8k^2$ triples, select 8 triples that all have different types as follows: within a set of $8k^2$ triples, inspect each triple in order and either *select it* or provide a proof that it is of the same type as a previously selected triple. If at the end, 8 triples have been selected, then either all 8 triples are of different types, or one type did not occur within the set at all. In the former case, prove that one of the selected triples has type $(0, 0, 0)$ and discard it. Denote the remaining 7 *selected* triples as $((\alpha_j^1, \beta_j^1, \gamma_j^1), \ldots, (\alpha_j^7, \beta_j^7, \gamma_j^7))$.

9. Finally, for each j, show that for some $1 \leq t \leq 7$, (a_j, b_j, c_j) is of the same type as $(\alpha_j^t, \beta_j^t, \gamma_j^t)$. Note, this proves that the clause is satisfied since the identified triple $(\alpha_j, \beta_j, \gamma_j)$ is not of type $(0,0,0)$.

In the following, we refer to the portion of π generated by step I in the honest prover algorithm as π_I. The single-theorem verifier algorithm, V, proceeds as follows.

1. Run the honest-prover algorithm as per step $1, 2, 4$ and 7 to generate π_2, π_4, π_7 and verify that the corresponding proof string parts are equivalent. Also verify that every root given in the proof string has Jacobi symbol 1 and is less than $n/2$. Reject if not.
2. As per [BSMP91], verify π_3, which is the proof that (x, y) is well-formed.
3. Verify π_5 by making sure that each pair is handled, and that the proof string contains a proper root of the pair.
4. Verify π_8 by checking that for each set of triples, the prover has handled the pairs in order, and that each of the proofs given between triples is sound. Finally, verify that the opened pair is of type $(0, 0, 0)$
5. For each clause, verify the proof that it is associated with one of it's remaining seven selected triples.

As in [BSMP91], we now transform the single theorem system to a multiple theorem one by breaking the random string into three pieces, ρ, τ_1 and τ_2. We use ρ to prove that (x_0, y_0) in a proper proving pair[6] . This is done exactly as in Step 3. At this point, x_0 and y_0 can be used with τ_2 to prove the first theorem as in the single theorem case (starting from Step 4 since the correctness of (x_0, y_0) has already been established).

At this point, our construction diverges from [BSMP91]. Originally, for the second theorem, the prover in [BSMP91] randomly selects completely new proving pairs (x_{00}, y_{00}) and (x_{01}, y_{01}) and then uses (x_0, y_0) and τ_1 along with the single theorem system to prove the auxiliary theorem, "(x_{00}, y_{00}) and (x_{01}, y_{01}) are properly formed proving pairs." [7] This approach, however, does not work in our setting because selecting new random values after posting the public key compromises the Uniqueness property.

To circumvent this difficulty, we add a seed, s, for a pseudo-random function f [GGM86] to the prover's secret key, and a perfectly binding commitment to s to the prover's public key. Now whenever the prover in [BSMP91] is instructed to prove that

> "$(x_{0b_1\ldots b_i 0}, y_{0b_1\ldots b_i 0})$ and $(x_{0b_1\ldots b_i 1}, y_{0b_1\ldots b_i 1})$ are properly formed proving pairs"

[6] We have changed notation from (x, y) above to (x_0, y_0) in order to match the notation from [BSMP91]

[7] In general, [BSMP91] describes a tree structure in which $(x_{0b_1\ldots b_i}, y_{0b_1\ldots b_i})$ is used to certify $(x_{0b_1\ldots b_i 0}, y_{0b_1\ldots b_i 0})$ and $(x_{0b_1\ldots b_i 1}, y_{0b_1\ldots b_i 1})$ which are then used to prove the $b_1 \ldots b_i 0^{\text{th}}$ and $b_1 \ldots b_i 1^{\text{th}}$ theorems.

our prover instead proves that

> "$(x_{0b_1\dots b_i0}, y_{0b_1\dots b_i0})$ and $(x_{0b_1\dots b_i1}, y_{0b_1\dots b_i1})$ are generated using the BDMP honest prover algorithm with coins $f_s(0b_1 \dots b_i)$"

Observe that this auxiliary theorem is an NP-statement whose length is a fixed polynomial in k and can therefore be proven using the single theorem uniZK system with a sufficiently long τ_1. This assures both that $(x_{0b_1\dots b_i0}, y_{0b_1\dots b_i0})$ and $(x_{0b_1\dots b_i1}, y_{0b_1\dots b_i1})$ have the necessary properties and also that the prover had no choice in selecting these values (given his public key).[8]

We can also extend our system to work for theorems of arbitrary size by using techniques similar to those in [BSMP91]. Let ϕ be an arbitrarily long formula and let $(\hat{x}, \hat{y})$ be the next proving pair in the tree construction described above. First, use $(\hat{x}, \hat{y})$ to complete steps 4 through 7. Observe that we cannot continue with step 8 because τ_2 is not long enough to accommodate all of the clauses of ϕ. Instead, for each clause, we form the NP-statement

In clause j of ϕ, the triple (a_j, b_j, c_j) contains one non-residue $\mod \hat{x}$.

Note that the length of this statement is fixed and independent of the size of ϕ. Therefore, by making τ_2 sufficiently long, we can prove each of these statement as separate theorems using the successor pairs of $(\hat{x}, \hat{y})$ as per the multi-theorem construction. Note that the prover has no choices to make since the form of the statement and the order in which they are proven are fixed by the statement ϕ.

Security Properties The proof that this scheme is complete, sound, and zero-knowledge closely follows the corresponding proofs in [BSMP91]. Therefore, we will only sketch a proof that our construction satisfies Uniqueness.

First, we consider Uniqueness in the single-theorem case. Define the secret key extraction function, $sk()$, to take in a proving pair $PK = (x, y)$ and return the factorization of x. We now observe that if PK is not properly constructed, then with overwhelmingly high probability over the choice of random string, the verifier will reject any proof (because of soundness in Step 3), and therefore $\Pi_{PK}(\sigma, \phi)$ will be empty and uniqueness is automatic.

Therefore, we restrict attention to the case when PK is properly formed. First we observe that P (with auxiliary inputs σ, ϕ and the factorization of x) is a deterministic function and that by completeness it maps W_x into $\Pi_{PK}(\sigma, \phi)$. We then put forward an efficient algorithm P^{-1} (with the same auxiliary inputs) and show that it is the inverse of P. Finally, we show P and P^{-1} are bijections by proving that P^{-1} is an injection.

In the following we refer to the portion of π generated by step I in the honest prover algorithm as π_I. Let P^{-1} on input $\pi \in \Pi_{PK}(\sigma, \phi)$ inspect the portion π_6, use the factorization of x to determine the quadratic character (mod x) of

[8] Note here that we need to use a commitment scheme with only a single valid decommit message (to assure that the prove does not have a choice in selecting the witness for the auxiliary theorem).

$u_1, \ldots, u_n$, and output the corresponding assignment w. Note by inspection of step 6 P^{-1} returns the exact assignment that was used to generate π, so P^{-1} is the inverse of P.

All that remains to be shown is that P^{-1} is injective. We do this by showing that if $\pi^* \neq \pi = P(\sigma, \phi, w, sk(PK))$ and yet $P^{-1}(\sigma, \phi, \pi^*, sk(PK)) = w$ then $\pi^* \notin \Pi_{PK}(\sigma, \phi)$. We establish this using case analysis. Suppose the first point at which π and π^* differ is portion π_I. For all cases, except for $I = 8$, the proof is straightforward based on the Verifier's algorithm.

For case $I = 8$, we first argue that the sub-proof used to show that two triples are of the same type is sound. This follows directly from the fact that (x, y) is properly formed.

We next show that π^* cannot select two triples of the same type. If π^* selects two triples of the same type, then some type, is not selected. With high probability, this unselected type appears in the set of $8k^2$ triples. Therefore, the Verifier rejects π^* since π^* cannot prove that the unselected type is similar to a previously selected triple. Hence, π^* must select all 8 types.

If π and π^* select the same 8 triples, then the fact that π^* is rejected follows from the fact that each quadratic residue has exactly one Jacobi symbol 1 root less than $x/2$.

Assume π and π^* select different triples. If π selects a triple that π^* does not, then π^* must give a false proof that this triple was the same as a previously selected one, and we already know that the Verifier rejects such proofs. Alternatively, if π^* selects a triple not selected by π, then π^* cannot contain 8 different types, and we know that the Verifier rejects in this case as well.

This completes our proof of uniqueness in the single-theorem case. The only difference in the multi-theorem case is that π and π^* might use different pairs $(x, y) \neq (x^*, y^*)$ to prove theorem i. This means that (x^*, y^*) is not the output of the honest prover algorithm with coins specified by the committed seed in the prover's public key. In this case, by the soundness of the single-theorem proof system, the verifier will reject any auxiliary proof certifying (x^*, y^*). $\square$

Remark: Choosing The Right NP-Complete Problem. We deliberately choose 3SAT (over, say, 3-Colorability) because, in order to satisfy the Uniqueness property, our multi-theorem construction requires a reduction from general NP-statements to 3-SAT formula which preserves the number of witnesses (in our case, one to one). Notice that even parsimonious reductions for 3-colorability map one witness to six possible colorings.

Remark: Choosing The Right Complexity Assumption. There are several NIZK systems based on the more general assumption that trap-door permutations exist (e.g., [FLS90] and [KP98]). Adapting such systems to admit Unique proofs, however, seems to require substantially new techniques.

4 Fair Zero-Knowledge Proofs

Informally, the goal of Fair ZK is to be a ZK proof system which remains secure even when the prover maliciously colludes with some subset of the verifiers.

This goal is embodied by the four properties of completeness, soundness, zero-knowledgeness, and fairness. Completeness states that if the prover and all verifiers are honest, than all true theorems are provable. Soundness states that even if a (computationally unbounded) dishonest prover collaborates with malicious verifiers during the set-up stage, no honest verifier will accept a false theorem. Zero knowledgeness states that even if all verifiers are malicious, they are unable to extract from the prover any extra information except that x_i is true and it is the ith theorem. Zero knowledgeness is formalized by the existence of an efficient simulator S that generates the same view that the malicious verifiers would have seen had they interacted with the *honest* prover about the same sequence of theorems (without seeing the corresponding witnesses). Importantly, S succeeds even if it is given each theorem one at a time (without knowing what future theorems might be). Fairness states that, *as long as an honest verifier accepts* all of the theorems, then, no matter how a dishonest prover might collude with a set of malicious verifiers, no verifier learns anything other than "x_i is true and it is the ith theorem." This is again formalized via a second simulator S^* that generates the same views that the malicious verifiers would have seen if they interacted with the *dishonest* prover. Again, S^* succeeds even though it is given the sequence of theorems one at at time. As far as we know, this is the first use of the simulator paradigm to protect the secrets of one dishonest party from another dishonest party.

Remarks

1. The primary difficulty with simulating a dishonest prover is that the prover has a witness, and the simulator does not! Clearly, if the prover decides to cheat and output the witness (or some partial information about it) in lieu of a valid proof, there is no hope for a simulator to produce indistinguishable transcripts. Thus, the best one can hope for is to require that simulated proofs are indistinguishable from real proofs *conditioned* on the event that an honest verifier accepts *all* the real proofs.
2. It is crucial to the applicability of Fair ZK that it applies to an unbounded sequence of theorems. And it is this feature that prevents us (at least for now) from relying on general cryptographic assumption. In particular, "single-theorem" Fair ZK can be achieved without number-theoretic assumptions by suitable modifying [DMP91].
3. In order to guarantee that no verifier gets additional knowledge about theorem x_i, an honest verifier must monitor all "utterances" of the prover as soon as he hands him an envelope in the preprocessing phase. In particular, the honest verifier must also monitor the first $i-1$ proofs : If all honest verifiers are "out to lunch", a dishonest prover may send sk to her accomplices!
4. The order in which a sequence of theorems is proven must be fixed. Giving the prover freedom to choose this order provides yet another opportunity for steganography. (Achieving Fair ZK requires us to run a tight ship!) However, the prover may receive all theorems and witnesses, if available, immediately after completing the setup protocol successfully.

4.1 Formal Definition

A *setup protocol* is a protocol, $(\mathcal{P}, \mathcal{V}_1, \ldots, \mathcal{V}_n)$, with a distinguished ITM $\mathcal{P}$, the prover (referred to as player 0), and n ITMs, $\mathcal{V}_1, \ldots, \mathcal{V}_n$, the verifiers (respectively referred to as players 1 through n). All players in this protocol exchange message via broadcast; in addition the verifiers may also send messages in envelopes and the prover also receives messages in envelopes. Each execution e of the setup protocol produces a common public output $pk \in \{0,1\}^* \cup \{\perp\}$ and a secret output sk for the prover.

In an execution e of this protocol with security parameter 1^k, we denote by $\textsc{view}_i(e)$ the triple $(1^k, \rho_i, M_i)$, where ρ_i is the random tape for player i and M_i is the set of messages received by player i during the execution. If $T = (a, b, \ldots)$ is a sequence of players, then denote by $\textsc{view}_T(e)$ the sequence of views $(\textsc{view}_a(e), \textsc{view}_b(e), \ldots)$.

We denote by $(pk, sk), e \leftarrow \langle \mathcal{P} \overset{1^k}{\longleftrightarrow} \mathcal{V}_1, \ldots, \mathcal{V}_n \rangle$ the random variable obtained by uniformly and independently selecting a random tape ρ_i for each player i, executing the setup protocol with security parameter 1^k and random tapes ρ_i's, and outputting the so generated execution e, with its corresponding outputs pk and sk.

Definition 1 (Fair Zero Knowledge) *Let L be a (computationally) unique-witness language*[9] *A* Fair zero-knowledge proof system *for L consists of (1) a* setup protocol, $(\mathcal{P}, \mathcal{V}_1, \ldots, \mathcal{V}_m)$*; (2) an efficient deterministic* proving algorithm P*, (3) an efficient verification algorithm V, and a negligible function, μ, such that the following properties are satisfied:*

Completeness $\forall$ *theorem-witness sequences* $(x_1, w_1), (x_2, w_2), \ldots$ *for L and* $\forall k \in Z^+$,

$$\Pr\left[\begin{array}{l} (pk, sk), e \leftarrow \langle P \overset{1^k}{\longleftrightarrow} V_1, \ldots, V_m \rangle; \\ \qquad\qquad \pi_1 \leftarrow P(x_1, w_1, sk, 1);\ \pi_2 \leftarrow P(x_2, w_2, sk, 2);\ \ldots \\ \qquad\qquad : \ \bigwedge_i V(x_i, \pi_i, pk, i) = 1 \end{array}\right] > 1 - \mu(k)$$

Soundness $\forall\ P^*, V_1^*, \ldots, V_{i-1}^*, V_{i+1}^*, \ldots, V_n^*$, $\forall$ *sufficiently large* $k \in Z^+$,

$$\Pr\left[\begin{array}{l} (pk^*, sk^*), e \leftarrow \langle P^* \overset{1^k}{\longleftrightarrow} V_1^*, \ldots, V_{i-1}^*, V_i, V_{i+1}^*, \ldots, V_n^* \rangle; \\ \quad (x^*, \pi^*, i) \leftarrow P^*(\textsc{view}_0(e)) \\ \qquad\qquad : \ x^* \notin L \ \wedge \ V(x^*, \pi^*, pk^*, i) = 1 \end{array}\right] < \mu(k)$$

[9] A *computationally unique-witness* language is one in which, given a witness w for a statement $x \in L$, it is hard to produce a new witness for the same statement.

Zero-Knowledgeness $\forall$ *efficient ITMs* $V_1^*, \ldots, V_n^*$, $\exists$ *an efficient algorithm* S *such that* $\forall$ *theorem-witness sequences* $(x_1, w_1), (x_2, w_2), \ldots$ *for* L,

$$\left\{ \begin{array}{c} (pk, \alpha, \text{VIEW}) \leftarrow S(1^k); \\ \pi_1 \leftarrow S(x_1, \alpha, 1); \\ \pi_2 \leftarrow S(x_2, \alpha, 2); \ldots : \\ pk, \text{VIEW}, x_1, \pi_1, x_2, \pi_2 \ldots \end{array} \right\}_k \overset{c}{\approx} \left\{ \begin{array}{c} (pk, sk), e \leftarrow \langle P \overset{1^k}{\longleftrightarrow} V_1^*, \ldots, V_n^* \rangle; \\ \pi_1 \leftarrow P(x_1, w_1, sk, 1); \\ \pi_2 \leftarrow P(x_2, w_2, sk, 2); \ \ldots : \\ pk, \text{VIEW}_{1,\ldots,n}(e), x_1, \pi_1, x_2, \pi_2 \ldots \end{array} \right\}_k$$

Fairness $\forall$ *efficient* $P^*, V_1^*, \ldots, V_{i-1}^*, V_{i+1}^*, \ldots, V_n^*$, $\exists$ *an efficient* S^* *such that* $\forall$ *theorem-witness sequences* $(x_1, w_1), (x_2, w_2), \ldots$ *for* L, *the following two ensembles are computationally indistinguishable:*

$$\left\{ \begin{array}{l} (pk^*, \alpha, \text{VIEW}) \leftarrow S^*(1^k); \\ \qquad \pi_1^* \leftarrow S^*(x_1, \alpha, 1); \ \pi_2^* \leftarrow S^*(x_2, \alpha, 2); \ldots \\ \qquad : \ pk^*, \text{VIEW}, x_1, \pi_1^*, x_2, \pi_2^* \ldots \end{array} \right\}_k$$

$$\left\{ \begin{array}{l} (pk^*, sk^*), e \leftarrow \langle P^* \overset{1^k}{\longleftrightarrow} V_1^*, \ldots, V_{i-1}^*, V_i, V_{i+1}^*, \ldots, V_n^* \rangle; \\ \pi_1^* \leftarrow P^*(x_1, w_1, sk^*, 1); \ \pi_2^* \leftarrow P^*(x_2, w_2, sk^*, 2); \ \ldots \\ : pk^*, \text{VIEW}_{1,\ldots,i-1,i+1,\ldots,n}(e), x_1, \pi_1^*, x_2, \pi_2^*, \ldots \end{array} \middle| \ \bigwedge_i V(x_i, \pi_i, pk^*, i) = 1 \right\}_k$$

4.2 Constructing Fair ZK

Our goal is to defeat steganographic attacks by using uniZK. However, we cannot allow the prover to pick his own secret key (since he might share it with a verifier beforehand). Therefore, we need to incorporate randomness from *all* of the verifiers during the selection of a prover secret key. We show that if we allow the prover to receive a *single envelope* from each verifier during the preprocessing, that we can transform any UniZK system into a Fair zero-knowledge proof system.

Theorem 2. *The existence of a* uniZK *proof system and a family of trap-door one-way permutations implies a fair zero knowledge proof system in which, during the preprocessing phase, each verifier sends the prover a single envelope.*

Proof Sketch: We first present our fair zero-knowledge protocol and then sketch a proof that the protocol satisfies the security properties specified in our definition of fair zero-knowledge.

- Preprocessing Phase:
  1. The players engage in a simulatable coin-flipping protocol which is secure against an unbounded Prover to generate a reference string, σ. For example, in order to generate a single bit, the prover uses a perfectly binding commitment scheme to commit to a random bit. Then all the verifiers broadcast (in turn) a commitment of a randomly chosen bit, then decommit the bits in the opposite order, and finally the prover decommits her bit. The output is defined as the xor of all opened bits. This can be repeated sequentially to generate longer reference strings.

2. The players partially execute the secure function evaluation protocol from [GMW87] with privacy threshold set to $n-1$ in order to compute the following n-valued function:

$$F(\varepsilon, \ldots, \varepsilon) = \{ \overset{\text{Prover's output}}{SK_{uzk}} , \overset{\text{Verifiers' outputs}}{PK_{uzk}, \ldots, PK_{uzk}} \}$$

   That is, the function produces a private output for the prover consisting of a uniZK secret key, and produces the corresponding uniZK public key as the output for all of the verifiers.
   The GMW protocol is executed until all of the players have shares of each of the output values, but have not yet sent each other these shares.
3. All of the shares for the verifiers' outputs are broadcast to all parties. Note that there is no need to encrypt these shares as all of the verifiers have the same output values. As in the original GMW protocol, all parties use interactive zero-knowledge proofs in order to prove to all other parties that the share they have broadcast is correctly computed.
4. In the final round all verifiers send their shares of the prover's output as well as all random coins that they used during preprocessing to the prover using an envelope channel.
5. The prover runs the honest verifier algorithm to verify that the shares sent by the verifiers were computed correctly. It then computes its private output, namely the SK_{uzk}, by combining the shares. At this point, the prover has unique knowledge of a uniZK secret key, and all parties have a corresponding uniZK public key and a reference string, σ.

– Proof Phase:
  In the proof phase, the prover can prove any number of theorems by using the uniZK prover algorithm with secret key SK_{uzk} and reference string, σ. The verifiers use the corresponding uniZK verifier algorithm with public key PK_{uzk} and random string σ to verify each proof. As soon as a single proof fails to verify, the verifiers are instructed to reject all subsequent proofs.

Security Properties The completeness of this system is straightforward. Therefore, we will only sketch proofs that our system satisfies Soundness, Zero-Knowledgeness and Fairness.

Soundness Here we must show that any prover who manages to cheat against a set of verifiers either breaks the correctness property of the coin flipping protocol, or breaks the soundness property of the uniZK system, both of which are unconditionally secure. Assume that the output from the coin-flipping protocol is truly random. In this case, the prover's algorithm for cheating can be used without modification to break the soundness of the uniZK system. Note, even if the Prover breaks the correctness of the SFE, thereby generating the uniZK keys of her choice, this does not allow the Prover to break soundness since the uniZK system is sound, even when the prover chooses his key after seeing the reference string.

Zero-knowledgeness Our simulator, S, works as follows:

1. First run the uniZK simulator in order to generate a reference string σ^* as well as a public and private key, PK_{uniZK} and SK_{uniZK}.
   The goal is to now manipulate the coin-flipping protocol and the secure function evaluation in order to produce σ^* and PK_{uniZK}.
2. Use the simulator for the coin-flipping protocol in order to generate a transcript with output σ^*.
3. Begin running the secure function evaluation protocol. At any point during which S is required to send a message on behalf of any party, write the message to the transcript as an honest party would.
4. During the last step when each party broadcasts its share of the public output and proves that it was formed correctly, S uses it ability to rewind the malicious parties in order to do two things. First, it learns the shares of each of the malicious parties by proceeding honestly. It then rewinds the malicious parties, and broadcasts shares on behalf of the honest parties to force the public output to be PK_{uniZK}. Finally, by rewinding, it simulates the zero-knowledge proofs that the broadcast shares are correct.
5. The envelopes that are sent from the malicious parties are opened and the random coins inside are used for verification. Upon failure, S aborts.
6. S now uses the uniZK simulator in order to generate the proofs for the sequence of theorems that arrive using its key SK_{uniZK} and σ^*.

In order to prove that the transcripts produced by this Simulator are indistinguishable from those of a real execution, we first note that the transcript for the coin-flipping protocol is generated by a simulator and thus indistinguishable. During the SFE portion of the protocol, all of the steps are identical until Step 4. During the last two steps, S is using another simulator to generate indistinguishable transcripts for a zero-knowledge proof. Therefore, any distinguisher of the Fair ZK protocol's transcripts can be trivially used to break the zero-knowledge property of the proof used in this step. Since the envelope traffic is not part of the view of the verifiers, it does not matter what is sent in them. Therefore, the verifiers have no information about the SK_{uniZK} since they have no information about the Prover's share. Therefore, any distinguisher between the key produced by S and the key produced in a real execution can be used to break the uniZK simulator.

Similarly, the proof strings are henceforth produced by the uniZK simulator and therefore any distinguisher can also be used (in a straightforward reduction) to break the uniZK simulator.

Fairness The same simulator used to prove the zero-knowledgeness property is also used to prove the Fairness property. The only difference is that the simulator must use the cheating prover algorithm in order to generate all of the prover messages during the preprocessing phase. Note that during preprocessing the simulator is able to directly run the malicious prover algorithm because until a witness is given to the prover, the prover has no secrets which the simulator does not know. This step ensures that any secret agreements between the cheating prover and any malicious set of verifiers reflect themselves during the simulated transcripts (and therefore maintain indistinguishability with real executions).

Once the envelopes are sent to the prover, the uniqueness property of the uniZK system guarantees that for each theorem either the prover gives the single acceptable uniZK proof (which can be simulated) or she sends any other string in which case the honest verifier algorithm rejects. In the former case, fairness is guaranteed by the indistinguishability of uniZK. In the later case, fairness is vacuous because an honest verifier rejects. □

Corollary 1 *Under the Quadratic Residuosity assumption, there exists a fair zero knowledge proof system consisting of a preprocessing phase during which each verifier sends the prover a single envelope.*

This result follows directly from Theorem 2 and Theorem 1.

References

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC 1988*, pages 103–112, 1988.

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

[Dam92] I. Damgard. Non-interactive circuit based proofs and non-interactive perfect zeroknowledge with preprocessing. In *EUROCRYPT '92*. Springer-Verlag, 1992.

[DGB87] Y. Desmedt, C. Goutier, and S. Bengio. Special uses and abuses of the Fiat-Shamir passport protocol. In *CRYPTO '87*. Springer-Verlag, 1987.

[DMP91] Alfredo DeSantis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge with preprocessing. In *CRYPTO 1988*. Springer-Verlag, 1991.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *Proc. 31th FOCS*, pages 308–317, 1990.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc of STOC '87*, pages 218–229. ACM, 1987.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28(2), 1984.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[KP98] Joe Kilian and Erez Petrank. An efficient noninteractive zero-knowledge proof system for np with general assumptions. *J. Cryptology*, 11(1):1–27, 1998.

[NHvA02] John Langford Nicholas Hopper and Luis von Ahn. Provably secure steganography. In *CRYPTO 2002*. Springer-Verlag, 2002.