Certifying Permutations: Noninteractive Zero-Knowledge Based on Any Trapdoor Permutation*

Mihir Bellare

Department of Computer Science & Engineering, Mail Code 0114,
University of California at San Diego, 9500 Gilman Drive,
La Jolla, CA 92093, U.S.A.
mihir@cs.ucsd.edu

Moti Yung

Research Division, IBM T.J. Watson Research Center,
PO Box 704, Yorktown Heights, NY 10598, U.S.A.
moti@watson.ibm.com

Communicated by Claude Crépeau

Abstract. In cryptographic protocols it is often necessary to verify/certify the "tools" in use. This work demonstrates certain subtleties in treating a family of trapdoor permutations in this context, noting the necessity to "check" certain properties of these functions. The particular case we illustrate is that of noninteractive zero-knowledge. We point out that the elegant recent protocol of Feige, Lapidot, and Shamir for proving NP statements in noninteractive zero-knowledge requires an additional certification of the underlying trapdoor permutation, and suggest a method for certifying permutations which fills this gap.

Key words. Zero-knowledge, Certification, Trapdoor permutations.

1. Introduction

Primitives such as the RSA function, the discrete log function, or, more generally, any trapdoor or one-way function, have applications over and above the "direct" ones to public-key cryptography. Namely, they are also (widely) used as "tools" in the construction of (often complex) cryptographic protocols. This paper points to the fact that in this second kind of application, some care must be exercised in the manner in which the "tool" is used. Checks might be necessary that are not necessary in public-key applications.

The need for such checks arises from the need to consider adverserial behavior of parties in a cryptographic protocol. Typically, the problem is that one cannot trust a party to "correctly" create the tool in question. For example, suppose a party A is supposed to give another party B a modulus N product of two primes, and an RSA exponent e, to specify an RSA function. On receipt of a number N and an exponent e, it might be important that the receiver know that e is indeed an RSA exponent (i.e., relatively prime to the Euler Phi function of N). This is because the use of RSA in the protocol might be such that making e not an RSA exponent could give A an advantage (such applications do exist). On the other hand, the secrecy of the prime factorization of N is probably important to A, and she is not prepared to yield the prime factors of N which would enable B to check the correctness of e by himself.[1]

Protocols address this issue in several ways. Often, they incorporate additional sub-protocols which "certify" that the "tool" used is indeed "correct." These subprotocols will usually need to be zero-knowledge ones. Since the "correctness" of the tool can usually be formulated as an NP assertion, such subprotocols can in some cases be realized, by using, say, the general interactive protocols of [GMW] and [BC] which enable any NP statement to be proven in zero-knowledge. However, not always. One reason is that these protocols yield only computational zero-knowledge, and we may be interested in stronger forms such as statistical. (One such example occurs in [BMO] where the use of the certified discrete log assumption is crucial.) Alternatively, as for the problem we are interested in here, we may want a noninteractive solution, so that again the above-mentioned general techniques are precluded.

The particular instance of this issue that we focus on in this paper is the use of trapdoor permutations in noninteractive zero-knowledge (NIZK) proofs. We point out that the elegant recent NIZK protocol of Feige *et al.* [FLS] makes the (implicit) assumption that the trapdoor permutation is "certified." We note that this assumption is not valid for standard (conjectured) trapdoor permutations like RSA or those of [BBS], and so their protocol cannot be instantiated with any known (conjectured) trapdoor permutation. We suggest a certification method to fill this gap, so that *any* trapdoor permutation truly suffices, and RSA or the construction of [BBS] may be used. Our certification method involves an NIZK proof that a function is "almost" a permutation, and might be of independent interest.

Below we begin by recalling the notions of trapdoor permutations and NIZK proofs. We then discuss the FLS protocol and indicate the source of the problem. We then, briefly, discuss our solution. Later sections specify the definitions and our solution in more detail.

1.1. *Trapdoor Permutations*

We begin by recalling, in some detail, the definition of a trapdoor permutation generator (we follow [BM]), and seeing what it means for such a generator to be certified.

---

[1] Such a problem is not present in public-key applications. If I wish to publish N and e to specify an RSA digitial signature scheme, there is no question of my incorrectly choosing e because it is not to my advantage to do so.

A *trapdoor permutation generator* is a triplet of polynomial-time algorithms (G, E, I) called the *generating*, *evaluating*, and *inverting* algorithms, respectively. The generating algorithm is probabilistic, and on input 1^n outputs a pair of n-bit strings $(f^*, \bar{f}^*)$, describing, respectively, a trapdoor permutation and its inverse. If x, y are n-bit strings, then so are $E(f^*, x)$ and $I(\bar{f}^*, y)$. Moreover, the maps $f, \bar{f} : \{0, 1\}^n \to \{0, 1\}^n$ specified by $f(x) = E(f^*, x)$ and $\bar{f}(y) = I(\bar{f}^*, y)$ are permutations of $\{0, 1\}^n$, and $\bar{f} = f^{-1}$. Finally, f is "hard to invert" without knowledge of $\bar{f}$. (We refer the reader to Section 2.2 for more precise definitions.)

Fix a trapdoor permutation generator (G, E, I). We call an n-bit string f^* a *trapdoor permutation* if there exists some n-bit string $\bar{f}^*$ such that the pair $(f^*, \bar{f}^*)$ has a nonzero probability of being obtained when we run G on input 1^n. It is important to note that not every n-bit string f^* is a trapdoor permutation. In fact, the set of n-bit strings which are trapdoor permutations may be a very sparse subset of $\{0, 1\}^n$, and perhaps not even recognizable in polynomial time. If it *is* recognizable in polynomial time, we say the generator is *certified*.

We note that efficent recognizability is a lot to ask for. Consider our two main (conjectured) examples of trapdoor permutation generators: RSA [RSA], and the factoring-based generator of Blum *et al.* [BBS]. Neither is likely to be certifiable. This is because, in both cases, certification would need the ability to recognize in polynomial time the class of integers which are a product of (exactly) two (distinct) primes.

The importance of certification arises, as will be seen, from the use of trapdoor permutations as "tools" in protocols. Typically, one party (for example, the prover) gives the other party (for example, the verifier) a string f^* which is supposed to be a trapdoor permutation. For security reasons he may not wish to reveal (as proof that it is indeed one) the string $\bar{f}^*$, but may nonetheless need to convince the verifier that f^* is indeed a trapdoor permutation. This is clearly easy if the underlying generator is certified. Otherwise, the protocol itself must address the task of giving suitable conviction that f^* is really a trapdoor permutation.

One-way versus trapdoor. Does the same certification issue arise also for one-way permutations? It depends on what we call one-way permutations. Two kinds of definitions are used. In the first, a one-way permutation is a single object, namely a map $f : \{0, 1\}^* \to \{0, 1\}^*$; since it is a single map, no certification is needed. However, candidate one-way permutations typically occur, like trapdoor permutations, as families; the best example is the discrete log, where a member of the family is specified by a prime p and a generator g. Now, the certification issue arises just as before. (For discrete log it is usually addressed by usage of the certified discrete log assumption. Here one also provides the prime factorization of $p - 1$, which enables one to check that g is indeed a generator.)

1.2. NIZK Proofs

The setting we focus on in this paper is that of NIZK proof systems. NIZK is an important notion for cryptographic systems and protocols which was introduced by Blum *et al.* [BFM], [BDMP]. There are numerous applications. In particular, Naor and Yung show how to use NIZK proofs to implement public-key cryptosystems secure against chosen-

ciphertext attack [NY], and Bellare and Goldwasser present a novel paradigm for digital signatures based on NIZK proofs [BG].

The model is as follows. The prover and verifier have a common input w and also share a random string (of length polynomial in the length of w). We call this string the *reference* string, and usually denote it by σ. The prover must convince the verifier of the membership of w in some fixed underlying NP language L. To this end, the prover is allowed to send the verifier a single message, computed as a function of w and σ (in the case where $w \in L$, we also give the prover, as an auxiliary input, a witness to the membership of w in L). We usually denote this message by p. The verifier (who is polynomial time) decides whether or not to accept as a function of w, σ and p. We ask that there exist a prover who can convince the verifier to accept $w \in L$, for all random strings σ (this is the *completeness* condition). We ask that for any prover, the probability (over the choice of σ) that the verifier may be convinced to accept when $w \notin L$ is small (this is the soundness condition). Finally, we ask the proof provided by the prover of the completeness condition (in the case $w \in L$) be zero-knowledge, by requiring the existence of an appropriate "simulator." For a more complete specification of what it means to be an NIZK proof system, we refer the reader to Section 2.3.

We focus here on protocols with efficient provers. That is, we want the prover of the completeness condition (we call it the "honest" prover) to run in polynomial (in $n = |w|$) time.

We note that we are considering what are called "single-theorem" or "bounded" NIZK proof systems. The primitive of importance in applications is the "many-theorem" proof system (see [BFM] and [BDMP]). However, the former is known to imply the latter, given the existence of one-way functions [DY], [FLS]. So we may, without loss of generality, stick to the former.

1.3. *The Need for Certification in the FLS Protocol*

Feige *et al.* [FLS] recently presented an elegant NIZK proof system based on the existence of trapdoor permutations. The assumption, implicit in their analysis, is that the underlying trapdoor permutation generator is certified. Here we indicate whence arises the need for this certification. Once we have identified the source of the problem, we will discuss how we propose to solve it.

Let L be a language in NP, and let (G, E, I) be a trapdoor permutation generator. Fix a common input $w \in \{0, 1\}^n$, and let σ denote the reference string. We describe how the prover and verifier are instructed to operate under the FLS protocol. First, however, we need some background and some notation.

First, note that even if f^* is not a trapdoor permutation, we may assume, without loss of generality, that $E(f^*, x)$ is n-bits long. Thus, f^* does specify (via E) a map from $\{0, 1\}^n$ to $\{0, 1\}^n$; specifically, the map given by $x \mapsto E(f^*, x)$. We call this map the function specified by f^* under E, and denote it by f. Of course, if f^* is a trapdoor permutation, then f is a permutation.

If x and r are n-bit strings then $H(x, r)$ denotes the dot product, over GF(2), of x and r (more precisely, $H(x, r) = \bigoplus_{i=1}^{n} x_i r_i$). The theorem of Goldreich and Levin [GL] says that H is a "hard-core" predicate for (G, E, I). Very informally, this means the following. Suppose we run G (on input 1^n) to get $(f^*, \bar{f}^*)$, select x and r at random

from $\{0, 1\}^n$, and let $y = f(x)$. Then, given y and r, the task of predicting $H(x, r)$, and the task of finding x, are equally hard.

We are now ready to describe the protocol.

The protocol first asks that the prover P run G on input 1^n to obtain a pair $(f^*, \bar{f}^*)$. P is then instructed to send f^* to V (while keeping $\bar{f}^*$ to himself).

The problem is right here, in this first step. The analysis of [FLS] assumes that the prover performs this step correctly. This may be justified under the assumption that the trapdoor permutation generator is certified. If the generator is not certified, a cheating prover could, when $w \notin L$, select, and send to the verifier, an n-bit string which is *not* a trapdoor permutation. As we will see, this could compromise the soundness of the protocol. We proceed.

Once the prover has supplied f^*, the reference string is regarded as a sequence $\sigma = y_1 r_1 \cdots y_l r_l$ of l blocks of size $2n$, where $l = l(n)$ is a (suitable) polynomial (block i consists of the pair of n bit strings $y_i r_i$). We say that the prover "opens block i with value b_i" if he provides the verifier with an n-bit string x_i such that $f(x_i) = y_i$ and $H(x_i, r_i) = b_i$. The prover now opens certain blocks of the random string (and the protocol specifies how an honest prover should choose which blocks to open). Based on the values of the opened blocks, their relative locations in the reference string, and the common input, the verifier decides whether or not to accept. Exactly how he does this is not relevant to our discussion. Exactly how the honest prover is supposed to decide which blocks to open (which he does as a function of the block, the common input, and his witness to the membership of the common input in L) is also not relevant to our discussion. What is important to note is that the soundness condition relies on the assumption that, with f^* fixed, there exists a *unique* way to open any given block. If it is possible for the prover to open a block with value either 0 or 1, then the soundness of the FLS protocol is compromised.

The assumption that there is (one and) only one way to open a block is justified if f^* is a trapdoor permutation, because, in this case, f is a permutation. However, if f^* is not a trapdoor permutation, then f may not be a permutation, and in such a case, the possibility exists that blocks may be opened with values of the prover's choice.

We note that the gap is not an academic one. Considering concrete cases, such as the use of RSA or the trapdoor permutations based on quadratic residuosity that are suggested by [BBS], we see that the prover may indeed cheat.

The solution that first suggests itself is that the prover prove (in NIZK) that he really got f by running the generator G (this is an NP statement). The problem is, however, that to prove this new statement itself requires the use of a trapdoor permutation, and we are only chasing our tail.

Note that in the above NIZK proof, a (cheating) prover may choose f^* as a function of the random string. However, as pointed out in [FLS], this causes no difficulties. We may assume, in the analysis, that the reference string is chosen after f^* is fixed; later we apply a simple transformation which results in the proof system being secure even if f^* was chosen as a function of σ. We will deal with this issue explicitly when it arises.

Feige *et al.* [FLS] also consider the case of a computationally unbounded prover, where they use a one-way, rather than trapdoor, permutation. As we saw above, the certification problem still arises just as before.

1.4. *Our Solution*

Let f^* denote the n-bit string provided by the prover in the first step of the FLS protocol, as described above. As that discussion indicates, soundness does not really require that f be a trapdoor permutation. All that it requires is that f be a permutation. So it would suffice to certify this fact.

To certify that a map from $\{0, 1\}^n$ to $\{0, 1\}^n$ is a permutation seems like a hard task (it is a coNP statement). What we will do is certify it is "almost" a permutation, and then show that this suffices.

More precisely, we call f an ε-permutation if at most an ε fraction of the points in $\{0, 1\}^n$ have more than one preimage under f. We show that on common input f^*, and access to a common (random) reference string of length $\varepsilon^{-1} \cdot n$, the prover can provide the verifier with a noninteractive, zero-knowledge proof that that f is an ε-permutation. For a more precise statement of the theorem and its proof we refer the reader to Section 3.

We then show that adding this step to augment a multitude of independent FLS protocol instances yields an NIZK proof system (for any NP language) given the existence of any (not necessarily certified) trapdoor permutation generator. A complete proof of this fact is in Section 4. We note that this proof is in fact quite independent of the details of the FLS protocol and can be understood without a deep knowledge of the techniques of that paper.

Our solution also applies for the usage of one-way (rather than trapdoor) permutations in the [FLS] protocol with an unbounded prover.

2. Preliminaries

We begin by summarizing some basic notation and conventions which are used throughout the paper. We then discuss trapdoor permutations and say what it means for them to be "certified." Finally, we recall the definition, and some basic properties, of NIZK proof systems.

2.1. *Notation and Conventions*

We use the notation and conventions for probabilistic algorithms that originated in [GMR].

We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write "$A(\cdot)$"; if it receives two we write "$A(\cdot, \cdot)$", and so on. If A is a probabilistic algorithm, then, for any input i, the notation $A(i)$ refers to the probability space which assigns to the string σ the probability that A, on input i, outputs σ.

If S is a probability space we denote its support (the set of elements of positive probability) by $[S]$.

If $f(\cdot)$ and $g(\cdot, \cdots)$ are probabilistic algorithms, then $f(g(\cdot, \cdots))$ is the probabilistic algorithm obtained by composing f and g (i.e., running f on g's output). For any inputs $x, y, \ldots$ the associated probability space is denoted $f(g(x, y, \ldots))$.

If S is a probability space, then $x \xleftarrow{R} S$ denotes the algorithm which assigns to x an element randomly selected according to S. In the case that $[S]$ consists of only one element e we might also write $x \leftarrow e$.

For probability spaces $S, T, \ldots,$ the notation

$$\Pr[p(x, y, \ldots): x \xleftarrow{R} S; y \xleftarrow{R} T; \ldots]$$

denotes the probability that the predicate $p(x, y, \ldots)$ is true after the (ordered) execution of the algorithms $x \xleftarrow{R} S, y \xleftarrow{R} T,$ etc.

Let f be a function. The notation

$$\{f(x, y, \ldots): x \xleftarrow{R} S; y \xleftarrow{R} T; \ldots\}$$

denotes the probability space which assigns to the string σ the probability

$$\Pr[\sigma = f(x, y, \ldots): x \xleftarrow{R} S; y \xleftarrow{R} T; \ldots].$$

When we say that a function is computable in polynomial time, we mean computable in time polynomial in the length of its first argument.

We are interested in families of efficiently computable functions of polynomial description. The following definition is a convenient way of capturing them.

Definition 2.1. Let $E(\cdot, \cdot)$ be a polynomial-time computable function. We say that E *specifies an efficiently computable family of functions* if for each $n > 0$ and each $f^*, x \in \{0, 1\}^n$ it is the case that $|E(f^*, x)| = n$. Let $n > 0$ and $f^* \in \{0, 1\}^n$. The *function specified by f^* under E* is the map from $\{0, 1\}^n$ to $\{0, 1\}^n$ given by $x \mapsto E(f^*, x)$.

2.2. *Trapdoor Permutations and Certified Ones*

We present a precise definition of trapdoor permutations and see what it means for them to be "certified." The definition that follows is from Bellare and Micali [BM].

Definition 2.2 (Trapdoor Permutation Generator). Let G be a probabilistic, polynomial-time algorithm, and let E, I be polynomial-time algorithms. We say that (G, E, I) is a *trapdoor permutation generator* if the following conditions hold:

- *Generation*: For every $n > 0$, the output of G on input 1^n is a pair of n bit strings.
- *Permutation*: For every $n > 0$ and $(f^*, \bar{f}^*) \in [G(1^n)]$, the maps $E(f^*, \cdot)$ and $I(\bar{f}^*, \cdot)$ are permutations of $\{0, 1\}^n$ which are inverses of each other (that is, $I(\bar{f}^*, E(f^*, x)) = x$ and $E(f^*, I(\bar{f}^*, y)) = y$ for all $x, y \in \{0, 1\}^n$).
- *Security*: For all probabilistic polynomial-time (adversary) algorithms $A(\cdot, \cdot, \cdot)$, for all constants c and sufficiently large n,

$$\Pr[E(f^*, x) = y: (f^*, \bar{f}^*) \xleftarrow{R} G(1^n); y \xleftarrow{R} \{0, 1\}^n; x \xleftarrow{R} A(1^n, f^*, y)] \leq n^{-c}.$$

We call G, E, I the *generating, evaluating,* and *inverting* algorithms, respectively.

The standard (conjectured) "trapdoor permutations," such as RSA [RSA] and the factoring-based ones of Blum *et al.* [BBS], do fit this definition, after some minor transformations (the need for these transformations arises from the fact that these number-theoretic functions have domain Z_N^* rather than $\{0, 1\}^n$; we refer the reader to [BM] for details).

If a trapdoor permutation generator (G, E, I) is fixed and $(f^*, \bar{f}^*) \in [G(1^n)]$ for some $n > 0$, then, in informal discussion, we call f^* a trapdoor permutation. It is important to note that not every n bit string f^* is a trapdoor permutation: it is only one if there exists some $\bar{f}^*$ such that $(f^*, \bar{f}^*) \in [G(1^n)]$. In fact, the set of ($n$ bit strings which are) trapdoor permutations may be a fairly sparse subset of $\{0, 1\}^n$, and, in general, may not be recognizable in polynomial (in n) time. If a trapdoor permutation generator *does* have the special property that it is possible to recognize a trapdoor permutation in polynomial time then we say that this generator is *certified*. The more formal definition follows.

Definition 2.3. Let (G, E, I) be a trapdoor permutation generator. We say that (G, E, I) is *certified* if the language

$$L_{G,E,I} = \bigcup_{n \geq 1} \{f^* \in \{0, 1\}^n : \exists \bar{f}^* \in \{0, 1\}^n \text{ such that } (f^*, \bar{f}^*) \in [G(1^n)]\}$$

is in BPP.

We note that standard (conjectured) trapdoor permutation generators are (probably) *not* certified. In particular, RSA is (probably) not certified, and nor is the trapdoor permutation generator of Blum *et al.* [BBS]. This is because, in both these cases, the (description of) the trapdoor permutation f^* includes a number which is a product of two primes, and there is (probably) no polynomial-time procedure to test whether or not a number is a product of two primes.

The importance of certification stems, as we have seen, from applications in which one party (for example, the prover) gives the other party (for example, the verifier) a string f^* which is supposed to be a trapdoor permutation. For security reasons he may not wish to reveal (as proof that it is indeed one) the string $\bar{f}^*$, but may nonetheless need to convince the verifier that f^* is indeed a trapdoor permutation. In particular, the (implicit) assumption in [FLS] is that the trapdoor permutation generator being used is certified. As the above indicates, this means that their scheme cannot be instantiated with RSA or the trapdoor permutations of [BBS]. In later sections we will show how to extend their scheme so that any (not necessarily certified) trapdoor permutation generator suffices (so that RSA or the generator of [BBS] may in fact be used).

We note that if (G, E, I) is a trapdoor permutation generator, $f^* \in \{0, 1\}^n$, and $x \in \{0, 1\}^n$, then we may assume, without loss of generality, that $E(f^*, x)$ is an n-bit string. Hence $E(f^*, \cdot)$ does specify some map from $\{0, 1\}^n$ to $\{0, 1\}^n$, even if f^* is not a trapdoor permutation. That is, in the terminology of Definition 2.1, we may assume, without loss of generality, that the algorithm E specifies an efficiently computable family of functions. Of course, the map $E(f^*, \cdot)$ need not be a permutation on $\{0, 1\}^n$.

2.3. NIZK Proof Systems

We consider NIZK proof systems for NP. It is helpful to begin with the following terminology.

Definition 2.4. Let $\rho(\cdot, \cdot)$ be a binary relation. We say that ρ is an NP-*relation* if it is polynomial-time computable and, moreover, there exists a polynomial p such that

$\rho(w, \bar{w}) = 1$ implies $|\bar{w}| \leq p(|w|)$. For any $w \in \{0, 1\}^*$ we let $\rho(w) = \{\bar{w} \in \{0, 1\}^*: \rho(w, \bar{w}) = 1\}$ denote the witness set of w. We let $L_\rho = \{w \in \{0, 1\}^*: \rho(w) \neq \emptyset\}$ denote the language defined by ρ. A *witness selector* for ρ is a map $W: L_\rho \to \{0, 1\}^*$ with the property that $W(w) \in \rho(w)$ for each $w \in L_\rho$.

Note that a language L is in NP iff there exists an NP-relation ρ such that $L = L_\rho$.

We recall the definition of computational indistinguishability of ensembles. First, recall that a function $\delta: \{0, 1\}^* \to \mathbb{R}$ is *negligible* if for every constant d there exists an integer n_d such that $\delta(w) \leq |w|^{-d}$ for all w of length at least n_d.

Definition 2.5. An *ensemble* indexed by $L \subseteq \{0, 1\}^*$ is a collection $\{E(w)\}_{w \in L}$ of probability spaces (of finite support), one for each $w \in L$. Let $\mathcal{E}_1 = \{E_1(w)\}_{w \in L}$ and $\mathcal{E}_2 = \{E_2(w)\}_{w \in L}$ be ensembles over a common index set L. We say that they are *(computationally) indistinguishable* if for every family $\{D_w\}_{w \in L}$ of nonuniform, polynomial-time algorithms, the function

$$\delta(w) \overset{\text{def}}{=} |\Pr[D_w(v) = 1: v \overset{R}{\leftarrow} E_1(w)] - \Pr[D_w(v) = 1: v \overset{R}{\leftarrow} E_2(w)]|$$

is negligible.

The definition that follows is based on that of Blum *et al.* [BDMP]. However, we state the zero-knowledge condition differently; specifically, we use the notion of a witness selector to state the zero-knowledge condition in terms of the standard notion of computational indistinguishability, whereas in [BDMP] the zero-knowledge condition makes explicit reference to "distinguishing" algorithms. The two formulations are, of course, equivalent (but we feel this one is a little simpler because of its "modularity").

Definition 2.6. Let ρ be an NP-relation and let $L = L_\rho$. Let P be a machine, let V be a polynomial-time machine, and let S be a probabilistic, polynomial-time machine. We say that (P, V, S) defines an *NIZK proof system* for ρ if there exists a polynomial $l(\cdot)$ such that the following three conditions hold:

- *Completeness*: For every $w \in L$ and $\bar{w} \in \rho(w)$,

$$\Pr[V(w, \sigma, p) = 1: \sigma \overset{R}{\leftarrow} \{0, 1\}^{l(n)}; p \leftarrow P(w, \bar{w}, \sigma)] = 1,$$

  where $n = |w|$.
- *Soundness*: For every machine $\hat{P}$ and every $w \notin L$,

$$\Pr[V(w, \sigma, p) = 1: \sigma \overset{R}{\leftarrow} \{0, 1\}^{l(n)}; p \leftarrow \hat{P}(w, \sigma)] \leq \tfrac{1}{2},$$

  where $n = |w|$.
- *Zero-knowledge*: Let W be any witness selector for ρ. Then the following two ensembles are (computationally) indistinguishable:
  (1) $\{S(w)\}_{w \in L}$.
  (2) $\{(\sigma, p): \sigma \overset{R}{\leftarrow} \{0, 1\}^{l(|w|)}; p \leftarrow P(w, W(w), \sigma)\}_{w \in L}$.

We call P the *prover*, V the *verifier*, and S the *simulator*. The polynomial l is the *length of the reference string*. We say that P is *efficient* if it is polynomial-time computable.

We call σ the "common random string" or the "reference string."

The choice of $1/2$ as the error-probability in the soundness condition is not essential. Given any polynomial $k(\cdot)$, the error-probability can be reduced to $2^{-k(n)}$ by running $k(n)$ independent copies of the original proof system in parallel and accepting iff all subproofs are accepting.

A stronger definition (see [BDMP]) asks that in the soundness condition the adversary $\hat{P}$ be allowed to select a $w \notin L$ as a function of the reference string. This definition is, however, implied by the one above. More precisely, given (P, V, S) satisfying the above definition, (P', V', S') satisfying the more stringent definition can be constructed by a standard trick. Hence, we stick to the simple definition.

We note we are considering what have been called "single-theorem" or "bounded" NIZK proof systems. That is, the given reference string can be used to prove only a single theorem. The primitive of importance in applications (see [BG] and [NY]) is the "many-theorem" proof system. However, De Santis and Yung [DY], and Feige *et al.* [FLS] have shown that the existence (for some NP-complete relation) of a bounded NIZK proof system with an efficient prover implies the existence (for any NP-relation) of a many-theorem NIZK proof system (with an efficient prover), as long as one-way functions exist. Hence, given that the (bounded) NIZK proof systems we construct do have efficient provers, we may, without loss of generality, stick to the bounded case.

3. An NIZK Proof that a Map Is Almost a Permutation

Suppose E specifies an efficiently computable family of functions (see Definition 2.1), and suppose $f^* \in \{0, 1\}^n$ for some $n > 0$. We address in this section the problem of providing an NIZK proof that the function specified by f^* under E is "almost" a permutation.

We note that although this problem is motivated by the need to fill the gap in the FLS protocol (see Section 1.3), the results of this section might be of interest in their own right. Thus, we prefer to view them independently, and make the link to [FLS] in the next section.

In addressing the task of providing an NIZK proof that the function specified by f^* under E is "almost" a permutation, we must begin by clarifying two things. First, we need to say what it means for a function $f: \{0, 1\}^n \to \{0, 1\}^n$ to be "almost" a permutation. Our definition, of an ε-permutation, follows. Second, we must also say what we mean, in this context, by an NIZK proof (because the problem is not one of language membership). This is clarified below and in the statement of Theorem 3.2.

We begin with the definition. It says that f is an ε-permutation if at most an ε fraction of the points in $\{0, 1\}^n$ have more than one preimage (under f). More formally, we have the following.

Definition 3.1. Let $n > 0$ and $f: \{0, 1\}^n \to \{0, 1\}^n$. The *collision set of f*, denoted $C(f)$, is $\{y \in \{0, 1\}^n: |f^{-1}(y)| > 1\}$. Let $\varepsilon \in [0, 1]$. We call f an *ε-permutation* if $|C(f)| \leq \varepsilon 2^n$.

We now turn to the NIZK proof. The formal statement and proof of the theorem follow. We begin, however, by saying, informally, what we achieve, and giving the idea.

We fix E specifying an efficiently computable family of functions, and we fix a map $\varepsilon: \{0, 1\}^* \to (0, 1]$. We consider a prover and verifier who share a (random) reference string and have as common input a string $f^* \in \{0, 1\}^n$. If f (the function specified by f^* under E) is a permutation, then the prover can convince the verifier to accept (this is the completeness condition). If f is not an $\varepsilon(n)$-permutation, then the verifier will usually reject (this is the soundness condition).

We note the gap between these two conditions: we are guaranteed nothing if f *is* an $\varepsilon(n)$-permutation (but not a permutation). This is one way in which this "proof system" differs from proofs of language membership, where there are only two possibilities: either the input is in the language (and completeness applies) or it is not (and soundness applies).

In addition, when f is a permutation, the interaction yields no (extra) knowledge to the verifier. This is formalized, as usual, by requiring the existence of an appropriate "simulator."

The idea is very simply stated. Let σ be the reference string, which we think of as divided into blocks of size n. If f is not an $\varepsilon(n)$-permutation, then each block has probability at most $1 - \varepsilon(n)$ of being in the range of f. So if we ask the prover to provide the inverse of f on $\varepsilon^{-1}(n)$ different blocks, then he can succeed with probability at most $(1 - \varepsilon(n))^{\varepsilon^{-1}(n)} \leq 1/2$. Moreover, a collection of preimages of f on random points provide no information about (the easily computed) f, so the proof is zero-knowledge.

Theorem 3.2. *Let E specify an efficiently computable family of functions. Let $\varepsilon: \mathsf{N} \to (0, 1]$, and assume ε^{-1} is polynomially bounded and polynomial-time computable. Then there is a polynomial-time oracle machine A, a polynomial-time machine B, and a probabilistic, polynomial-time machine M such that the following three conditions hold:*

- Completeness: *Let $n > 0$ and $f^* \in \{0, 1\}^n$. Let f denote the function specified by f^* under E. Suppose f is a permutation. Then*

$$\Pr[B(f^*, \sigma, p) = 1: \sigma \xleftarrow{R} \{0, 1\}^{\varepsilon^{-1}(n) \cdot n}; p \leftarrow A^{f^{-1}}(f^*, \sigma)] = 1.$$

   Here $A^{f^{-1}}$ denotes A with oracle f^{-1}.
- Soundness: *Let $n > 0$ and $f^* \in \{0, 1\}^n$. Let f denote the function specified by f^* under E. Suppose f is not an $\varepsilon(n)$-permutation. Then, for any function $\hat{P}$,*

$$\Pr[B(f^*, \sigma, p) = 1: \sigma \xleftarrow{R} \{0, 1\}^{\varepsilon^{-1}(n) \cdot n}; p \leftarrow \hat{P}(f^*, \sigma)] \leq \tfrac{1}{2}.$$

- Zero-knowledge: *Let $n > 0$ and $f^* \in \{0, 1\}^n$. Let f denote the function specified by f^* under E, and suppose f is a permutation. Then the distributions $M(f^*)$ and $\{(\sigma, p): \sigma \xleftarrow{R} \{0, 1\}^{\varepsilon^{-1}(n) \cdot n}; p \leftarrow A^{f^{-1}}(f^*, \sigma)\}$ are equal.*

Proof. We specify the algorithm for the verifier. Let $f^* \in \{0, 1\}^n$ and let $\sigma = \sigma_1 \cdots \sigma_{\varepsilon^{-1}(n)}$ where each σ_i has length n. Let f denote the function specified by f^* under E. On input f^*, σ, and a string p, the verifier B rejects if the length of p is not

$\varepsilon^{-1}(n) \cdot n$. Otherwise, it partitions p into consecutive blocks of size n. We denote the ith block by p_i, so that $p = p_1 \cdots p_{\varepsilon^{-1}(n)}$. Then B accepts iff for each $i = 1, \ldots, \varepsilon^{-1}(n)$ it is the case that $f(p_i) = \sigma_i$.

Next we specify the prover A. Let $f^* \in \{0, 1\}^n$ and let $\sigma = \sigma_1 \cdots \sigma_{\varepsilon^{-1}(n)}$ where each σ_i has length n. Let f denote the function specified by f^* under E, and suppose f is a permutation. On input f^* and σ, and given f^{-1} as oracle, A sets $p_i = f^{-1}(\sigma_i)$ for each $i = 1, \ldots, \varepsilon^{-1}(n)$. It then sets $p = p_1 \cdots p_{\varepsilon^{-1}(n)}$ and outputs p. It is easy to see that the completeness condition is true.

We now check the soundness condition. Let $f^* \in \{0, 1\}^n$ and let f denote the function specified by f^* under E. We recall that $C(f) = \{y \in \{0, 1\}^n : |f^{-1}(y)| > 1\}$ is the collision set of f. Let $D(f) = \{y \in \{0, 1\}^n : |f^{-1}(y)| = 0\}$ be the set of n bit strings not in the range of f. Note that $|D(f)| \geq |C(f)|$. We let $\delta(n) \overset{\text{def}}{=} |D(f)|/2^n$ denote the density of $D(f)$. Now assume f is not an $\varepsilon(n)$-permutation. Then $|C(f)| \geq \varepsilon(n)2^n$, and thus $\delta(n) \geq \varepsilon(n)$. For any fixed string $\sigma = \sigma_1 \cdots \sigma_{\varepsilon^{-1}(n)}$, the following are clearly equivalent:

- There exists a string p such that $B(f^*, \sigma, p) = 1$.
- For each $i = 1, \ldots, \varepsilon^{-1}(n)$ it is the case that σ_i is in the range of f.

However, if σ is chosen at random, then, for each $i = 1, \ldots, \varepsilon^{-1}(n)$, the probability that σ_i is in the range of f is at most $1 - \delta(n)$, independently for each i. So, for any $\hat{P}$,

$$
\begin{aligned}
\Pr[B(f^*, \sigma, p) = 1 : \sigma \overset{R}{\leftarrow} \{0, 1\}^{\varepsilon^{-1}(n) \cdot n} : p \leftarrow \hat{P}(f^*, \sigma)] &\leq [1 - \delta(n)]^{\varepsilon^{-1}(n)} \\
&\leq [1 - \varepsilon(n)]^{\varepsilon^{-1}(n)} \\
&\leq \tfrac{1}{2}.
\end{aligned}
$$

We now specify M. Let $f^* \in \{0, 1\}^n$ and let f denote the function specified by f^* under E. Suppose f is a permutation. On input f^*, the machine M picks $\tau_1, \ldots, \tau_{\varepsilon^{-1}(n)} \in \{0, 1\}^n$ at random and sets $\sigma_i = f(\tau_i)$, for each $i = 1, \ldots, \varepsilon^{-1}(n)$. It sets $p = \tau_1 \cdots \tau_{\varepsilon^{-1}(n)}$ and outputs (σ, p). The zero-knowledge is easy to check. $\square$

We note that, in the above, we are thinking of f^* as being the common input, and the reference string is chosen at random independently of f^*. Of course, in our application, the prover may choose f^* as a function of the reference string. This, however, is easily dealt with by a standard trick, and so, for the moment, we focus on the case presented here. When we put everything together (see Theorem 4.4) we will return to this issue and show explicitly how to deal with it, given what we establish here.

We note also that no cryptographic assumptions were needed for the above proof, and the zero-knowledge is "perfect."

4. Using the Certification Procedure

In this section we show how the certification procedure of Theorem 3.2 can be combined with the results of [FLS] to yield an NIZK proof system for any NP-relation. We stress that the argument we present here depends little on the specifics of the protocol of [FLS],

and our proof does not presume familiarity with that paper. We begin by extending
Definition 3.1 with the following terminology.

Definition 4.1.   Let $n > 0$ and $f\colon \{0, 1\}^n \to \{0, 1\}^n$. Let $\sigma = \sigma_1 \cdots \sigma_l$ for some $l \in \mathsf{N}$,
where each σ_i has length n. We say that σ *is f-bad* if there is an $i \in \{1, \dots, l\}$ such that
$\sigma_i \in C(f)$. We denote by $C_l(f)$ the set of all ln-bit strings which are f-bad.

We now state, without proof, a lemma which can be derived from [FLS]. The formal
statement follows, but, since it is rather long, we first try to give an informal explanation
of what it says.

Briefly, we show how to "measure" the "additional" error incurred by the [FLS]
protocol in the case where the function being used is not a permutation. More precisely,
we fix a trapdoor permutation generator (G, E, I) and an NP-relation ρ. In order to make
explicit the role played by the function used in the proof, we consider an interaction in
which the common input is a pair (w, f^*) of n-bit strings. The prover wishes to convince
the verifier that $w \in L \stackrel{\text{def}}{=} L_\rho$, using f^* as a "tool." We do not, *a priori*, know whether
or not f^* is a trapdoor permutation.

The completeness condition (below) says that if $w \in L$, then, assuming f^* really
is a trapdoor permutation, the prover can convince the verifier that $w \in L$. Moreover,
the zero-knowledge condition says this proof is zero-knowledge. The part we are really
concerned with, however, is the soundness condition.

The soundness condition says that if $w \notin L$, then the probability that a prover can
convince the verifier to accept is bounded by a small error $(1/4)$ plus a quantity that
depends on f^*. Specifically, this quantity is the probability that the reference string is
f-bad (see Definition 4.1), where f is the function specified by f^* under E.

A priori, this quantity may be large. Once we have stated the lemma, we show how to
use the results of the previous section to decrease it.

Lemma 4.2.   *Let (G, E, I) be a trapdoor permutation generator. Let ρ be an NP-
relation, and let $L = L_\rho$. Then there exists a polynomial-time machine $\bar{A}$, a polynomial-
time machine $\bar{B}$, a probabilistic, polynomial-time machine $\bar{M}$, and a polynomial $l(\cdot)$
such that the following three conditions hold:*

- Completeness: *For every $w \in L$, every $\bar{w} \in \rho(w)$, and every $(f^*, \bar{f}^*) \in [G(1^n)]$,*

$$\Pr[\bar{B}(w, \sigma, f^*, p) = 1\colon \sigma \stackrel{R}{\leftarrow} \{0, 1\}^{l(n)\cdot n}; p \leftarrow A(w, \bar{w}, \sigma, f^*, \bar{f}^*)] = 1,$$

  where $n = |w|$.
- Soundness: *For every machine $\hat{P}$, every $w \notin L$, and every $f^* \in \{0, 1\}^n$,*

$$\Pr[\bar{B}(w, \sigma, f^*, p) = 1\colon \sigma \stackrel{R}{\leftarrow} \{0, 1\}^{l(n)\cdot n}; p \leftarrow \hat{P}(w, \sigma, f^*)]$$
$$\leq \tfrac{1}{4} + \Pr[\sigma \in C_{l(n)}(f)\colon \sigma \stackrel{R}{\leftarrow} \{0, 1\}^{l(n)\cdot n}],$$

  where $n = |w|$ and f denotes the function specified by f^* under E.

- Zero-knowledge: *Let W be any witness selector for ρ. Then the following two ensembles are (computationally) indistinguishable:*

  (1) $\{(\sigma, f^*, p)\colon (f^*, \bar{f}^*) \xleftarrow{R} G(1^{|w|}); (\sigma, p) \xleftarrow{R} \bar{M}(w, f^*, \bar{f}^*)\}_{w \in L}$.

  (2) $\{(\sigma, f^*, p)\colon \sigma \xleftarrow{R} \{0, 1\}^{l(|w|) \cdot |w|}; (f^*, \bar{f}^*) \xleftarrow{R} G(1^{|w|});$
      $\quad p \leftarrow \bar{A}(w, W(w), \sigma, f^*, \bar{f}^*)\}_{w \in L}$.

Although we do not want to prove this lemma, some intuition is easily provided. Namely, [FLS] show that error probability less than any ε (in particular $\varepsilon = 1/4$) can be achieved, assuming only that the reference string is not f-bad (i.e., they do not use anywhere that f is permutation). Hence the prover can cheat with probability at most $1/4$ plus the probability that the reference string is f-bad. For the full proof, we refer the reader to [FLS].

We now show how to remove this extra f^* dependent term in the soundness condition by having the prover certify (using the proof system of Theorem 3.2) that f is almost a permutation. The lemma that follows provides the formal statement and proof, but we first say, informally, what is happening.

On common input (w, f^*), we have the prover give the proof of Lemma 4.2, and also, using a separate part of the reference string, run the procedure of Theorem 3.2. The verifier accepts iff both of these proofs are accepted (by their respective verifiers). The completeness and zero-knowledge conditions stay the same as in Lemma 4.2 (except that the reference string is longer, indicated by using a different symbol for its length); clearly, this is because the additional proof cannot hurt them. The soundness condition, however, now becomes more like a "real" soundness condition in that the "extra" term of Lemma 4.2 has disappeared.

In the proof of the new soundness condition, we have to consider two cases. First, we assume that f is "almost" a permutation, and show that in this case the "extra" term from the soundness condition of Lemma 4.2 is small. Second, we assume that f is not "almost" a permutation, and use the fact that we are guaranteed rejection (with high probability) by the soundness condition of Theorem 3.2.

Lemma 4.3. *Let (G, E, I) be a trapdoor permutation generator. Let ρ be an NP-relation and let $L = L_\rho$. Then there exists a polynomial-time machine A', a polynomial-time machine B', a probabilistic, polynomial-time machine M', and a polynomial $m(\cdot)$ such that the following three conditions hold:*

- Completeness: *For every $w \in L$, every $\bar{w} \in \rho(w)$, and every $(f^*, \bar{f}^*) \in [G(1^n)]$,*

$$\Pr[B'(w, \sigma, f^*, p) = 1\colon \sigma \xleftarrow{R} \{0, 1\}^{m(n) \cdot n}; p \leftarrow A(w, \bar{w}, \sigma, f^*, \bar{f}^*)] = 1,$$

  where $n = |w|$.

- Soundness: *For every machine $\hat{P}$, every $w \notin L$, and every $f^* \in \{0, 1\}^n$,*

$$\Pr[B'(w, \sigma, f^*, p) = 1\colon \sigma \xleftarrow{R} \{0, 1\}^{m(n) \cdot n}; p \leftarrow \hat{P}(w, \sigma, f^*)] \leq \tfrac{1}{2},$$

  where $n = |w|$.

- Zero-knowledge: *Let W be any witness selector for ρ. Then the following two ensembles are (computationally) indistinguishable:*

    (1) $\{(\sigma, f^*, p): (f^*, \bar{f}^*) \xleftarrow{R} G(1^{|w|}); (\sigma, p) \xleftarrow{R} M'(w, f^*, \bar{f}^*)\}_{w \in L}.$

    (2) $\{(\sigma, f^*, p): \sigma \xleftarrow{R} \{0, 1\}^{m(|w|) \cdot |w|}; (f^*, \bar{f}^*) \xleftarrow{R} G(1^{|w|});$
        $p \leftarrow A'(w, W(w), \sigma, f^*, \bar{f}^*)\}_{w \in L}.$

Proof. Let $\bar{A}, \bar{B}, \bar{M}$ be the machines, and let l be the polynomial, specified by Lemma 4.2. Let $\varepsilon(\cdot) = 1/(4l(\cdot))$. We apply Theorem 3.2 (with the algorithm E being the evaluating algorithm of our trapdoor family) to get a triplet of machines A, B, M satisfying the conditions of that theorem. We let $m(\cdot) = \varepsilon^{-1}(\cdot) + l(\cdot) = 5l(\cdot)$.

Notation. If σ is a string of length $m(n) \cdot n$, then $\sigma[1]$ denotes the first $\varepsilon^{-1}(n) \cdot n = 4l(n) \cdot n$ bits and $\sigma[2]$ denotes the last $l(n) \cdot n$ bits.

We now specify the algorithm for the verifier B'. Let $f^* \in \{0, 1\}^n$ and let σ be a string of length $m(n) \cdot n$. On input f^*, σ, and a string p, the verifier B rejects if $|p| < \varepsilon^{-1}(n) \cdot n$. Otherwise, it accepts if and only if

$$B(f^*, \sigma[1], p[1]) = 1 \quad \text{and} \quad \bar{B}(w, \sigma[2], f^*, p[2]) = 1,$$

where $p[1]$ denotes the first $\varepsilon^{-1}(n) \cdot n$ bits of p and $p[2]$ denotes the rest.

Next we specify A'. Let $w \in L$ and $\bar{w} \in \rho(w)$. Let $n = |w|$. Let $(f^*, \bar{f}^*) \in [G(1^n)]$. Let σ be a string of length $m(n) \cdot n$. On input $w, \bar{w}, \sigma, f^*, \bar{f}^*$, the machine A' sets $p[1] = A^{f^{-1}}(f^*, \sigma[1])$ (note that A' can obtain this output in polynomial time because, using $\bar{f}^*$, it can compute f^{-1} in polynomial time). It then sets $p[2] = \bar{A}(w, \bar{w}, \sigma[2], f^*, \bar{f}^*)$. Finally it sets $p = p[1]p[2]$ and outputs p. The fact that the completeness condition holds follows from the respective completeness conditions of Lemma 4.2 and Theorem 3.2.

Now for the interesting part, namely the soundness condition. Suppose $w \notin L$. Let $n = |w|$ and let $f^* \in \{0, 1\}^n$. Let f denote the function specified by f^* under E. We split the proof into two cases.

Case 1: f is a $\varepsilon(n)$-permutation. By assumption, $|C(f)| \le \varepsilon(n)2^n$. So

$$\Pr[\sigma[2] \in C_{l(n)}(f): \sigma[2] \xleftarrow{R} \{0, 1\}^{l(n) \cdot n}] \le \varepsilon(n)l(n) = \tfrac{1}{4}.$$

By the soundness condition of Lemma 4.2 it follows that, for every machine $\hat{P}$,

$$\Pr[\bar{B}(w, \sigma, f^*, p[2]) = 1: \sigma[2] \xleftarrow{R} \{0, 1\}^{l(n) \cdot n}; p[2] \leftarrow \hat{P}(w, \sigma, f^*)] \le \tfrac{1}{4} + \tfrac{1}{4} = \tfrac{1}{2}.$$

The soundness condition follows from the definition of B'. We proceed to the next case.

Case 2: f is not a $\varepsilon(n)$-permutation. The soundness condition of Theorem 3.2 implies that, for any function $\hat{P}$,

$$\Pr[B(f^*, \sigma[1], p[1]) = 1: \sigma[1] \xleftarrow{R} \{0, 1\}^{\varepsilon^{-1}(n) \cdot n}; p \leftarrow \hat{P}(f^*, \sigma[1])] \le \tfrac{1}{2}.$$

The soundness condition then follows directly from the definition of B'. This completes the proof of the soundness condition.

The zero-knowledge, again, follows immediately from Lemma 4.2 and Theorem 3.2. Let $w \in L$ and let $n = |w|$. Let $(f^*, \bar{f}^*) \in [G(1^n)]$. On input $w, f^*, \bar{f}^*$, machine M' runs M on input f^* to get an output $(\sigma[1], p[1])$. It then runs $\bar{M}$ on input $w, f^*, \bar{f}^*$ to get an output $(\sigma[2], p[2])$. It sets $\sigma = \sigma[1]\sigma[2]$ and $p = p[1]p[2]$ and outputs (σ, p).   □

One more step is needed to derive from Lemma 4.3 the existence of NIZK proof systems for any NP-relation (given the existence of a trapdoor permutation generator). Namely, the interaction must be on input w (alone); the prover must be allowed to select f^* (which in Lemma 4.3 is part of the common input) not only as a function of w but also as a function of the reference string. Clearly, in the completeness condition, we may simply ask the prover to select f^* by running the generation algorithm G. Any problems that arise will be in the soundness condition, where a cheating prover will take full advantage of the freedom to choose f^* as a function of the reference string.

For $w \notin L$, we may use the following "trick" (a standard probabilistic one, used, for the same purpose, in [BDMP] and [FLS]). For each fixed $f^* \in \{0, 1\}^n$, we reduce the probability that the verifier accepts the interaction on inputs (w, f^*) to $2^{-(n+1)}$, by parallel repetition. It follows that the probability that there exists a string $f^* \in \{0, 1\}^n$ such that the verifier accepts on input (w, f^*) is at most $2^n \cdot 2^{-(n+1)} = 1/2$. Details are below.

Theorem 4.4. *Let ρ be an NP-relation. Suppose there exists a trapdoor permutation generator. Then ρ possesses an NIZK proof system with an efficient prover.*

Proof. Let (G, E, I) be a trapdoor permutation generator. Let A', B', M' be the machines, and let m be the polynomial, specified by Lemma 4.3. Let $l(n) = m(n) \cdot n(n+1)$. We construct P, V, S satisfying the conditions of Definition 2.6.

Notation. If σ is a string of length $l(n)$, then we think of it as partitioned into $n + 1$ blocks, each of length $m(n) \cdot n$, and denote the ith block by $\sigma[i]$ $(i = 1, \ldots, n + 1)$.

We may assume, without loss of generality, that there is a polynomial t such that $B'(w, \cdot, \cdot, p) = 1$ only if $|p| = t(|w|)$. Let $L = L_\rho$. We specify V. Let $w \in L$ and $\sigma \in \{0, 1\}^{l(n)}$. On input w, σ, p, machine V rejects if $|p| \neq n + (n + 1)t(n)$. Otherwise, it sets f^* to the first n bits of p and p' to the rest. It further sets $p'[i]$ to the ith $t(n)$-bit block of p' $(i = 1, \ldots, n + 1)$. Now V accepts iff for each $i = 1, \ldots, n + 1$ it is the case that $B'(w, \sigma[i], f^*, p'[i]) = 1$.

We now specify P. Let $w \in L$ and $\bar{w} \in \rho(w)$. Let $n = |w|$, and let $\sigma \in \{0, 1\}^{l(n)}$. P runs G to obtain a (random) pair $(f^*, \bar{f}^*) \in [G(1^n)]$. It sets

$$p'[i] = A'(w, \bar{w}, \sigma[i], f^*, \bar{f}^*)$$

for $i = 1, \ldots, n + 1$, and sets $p' = p[1] \cdots p[n + 1]$. Finally it sets $p = f^* \cdot p'$ ("." denotes concatenation) and outputs p. The completeness condition (as required by Definition 2.6) follows from the completeness condition of Lemma 4.3.

Next we check the soundness condition. Suppose $w \notin L$. Let $n = |w|$ and let $f^* \in \{0, 1\}^n$. Let $\sigma \in \{0, 1\}^{l(n)}$. We say that σ is f^*-bad if there exists an $i \in \{1, \ldots, n + 1\}$

and a string $q \in \{0, 1\}^{t(n)}$ such that $B'(w, \sigma[i], f^*, q) = 1$. The soundness condition of Lemma 4.3 implies that

$$\Pr[\sigma \text{ is } f^*\text{-bad}: \sigma \xleftarrow{R} \{0, 1\}^{l(n)}] \leq 2^{-(n+1)}.$$

Now we say that a string $\sigma \in \{0, 1\}^{l(n)}$ is bad if there exists an n-bit string f^* such that σ is f^*-bad. It follows that

$$\Pr[\sigma \text{ is bad}: \sigma \xleftarrow{R} \{0, 1\}^{l(n)}] \leq 2^n \cdot 2^{-(n+1)} = \tfrac{1}{2}.$$

This implies the soundness condition (as required by Definition 2.6).

Finally, we specify the simulator. Let $w \in L$ and let $n = |w|$. On input w, the simulator S runs G on input 1^n to obtain a (random) pair $(f^*, \tilde{f}^*) \in [G(1^n)]$. For $i = 1, \ldots, n+1$ it runs M' on input w, f^*, $\tilde{f}^*$ to get an output $(\sigma[i], p'[i])$. It sets $\sigma = \sigma[1] \cdots \sigma[n+1]$ and $p' = p'[1] \cdots p'[n + 1]$. It then sets $p = f^*.p'$ and outputs (σ, p). The zero-knowledge (as required by Definition 2.6) can be argued based on the zero-knowledge condition of Lemma 4.3. We omit the details.   □

In particular, NIZK proof systems are constructible based on RSA.

Combining Theorem 4.4 with the result of [NY] yields the following.

Corollary 4.5. *Suppose there exists a trapdoor permutation generator. Then there exists an encryption scheme secure against chosen-ciphertext attack.*

Similarly, combining Theorem 4.4 with the result of [BG] yields the following.

Corollary 4.6. *Suppose there exists a trapdoor permutation generator. Then there exists an implementation of the signature scheme of [BG].*

References

[BG] M. Bellare and S. Goldwasser. New Paradigms for Digital Signatures and Message Authentication Based on Non-Interactive Zero-Knowledge Proofs. *Advances in Cryptology—Crypto 89 Proceedings*, Lecture Notes in Computer Science, Vol. 435, G. Brassard, ed., Springer-Verlag, Berlin, 1989.

[BM] M. Bellare and S. Micali. How To Sign Given any Trapdoor Permutation. *Journal of the Association for Computing Machinery*, Vol. 39, No. 1, January 1992, pp. 214–233.

[BMO] M. Bellare, S. Micali, and R. Ostrovsky. The True Complexity of Statistical Zero-Knowledge. *Proceedings of the 22nd ACM Annual Symposium on Theory of Computing*, 1990.

[BBS] L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing*, Vol. 15, No. 2, May 1986, pp. 364–383.

[BDMP] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-Interactive Zero-Knowledge Proof Systems, *SIAM Journal on Computing*, Vol. 20, No. 6, December 1991, pp. 1084–1118.

[BFM] M. Blum, P. Feldman, and S. Micali, Non-Interactive Zero-Knowledge Proof Systems and Applications, *Proceedings of the 20th ACM Annual Symposium on Theory of Computing*, 1988.

[BC] G. Brassard and C. Crépeau. Non-Transitive Transfer of Confidence: a perfect Zero-Knowledge Interactive Protocol for SAT and Beyond. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.

[DY] A. De Santis and M. Yung. Cryptographic Applications of the Metaproof and Many-prover Systems. *Advances in Cryptology—Crypto 90 Proceedings*, Lecture Notes in Computer Science, Vol. 537, A. J. Menezes and S. Vanstone, eds., Springer-Verlag, Berlin, 1990.

[FLS] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Based on a Single Random String. *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, 1990.

[GL] O. Goldreich and L. Levin. A Hard-Core Predicate for All One-Way Functions. *Proceedings of the 21st ACM Annual Symposium on Theory of Computing*, 1989.

[GMR] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, Vol. 17, No. 2, April 1988, pp. 281–308.

[GMW] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Design. *Journal of the Association for Computing Machinery*, Vol. 38, No. 1, July 1991, pp. 691–729.

[NY] M. Naor and M. Yung. Public-Key Cryptosystems Secure Against Chosen-Ciphertext Attacks. *Proceedings of the 22nd ACM Annual Symposium on Theory of Computing*, 1990.

[RSA] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, Vol. 21, No. 2, February 1978, pp. 120–26.