SIAM J. COMPUT.
Vol. 20, No. 6, pp. 1084–1118, December 1991

007

NONINTERACTIVE ZERO-KNOWLEDGE*

MANUEL BLUM†, ALFREDO DE SANTIS‡, SILVIO MICALI§, AND GIUSEPPE PERSIANO¶

Abstract. This paper investigates the possibility of disposing of interaction between prover and verifier in a zero-knowledge proof if they share beforehand a short random string.

Without any assumption, it is proven that noninteractive zero-knowledge proofs exist for some number-theoretic languages for which no efficient algorithm is known.

If deciding quadratic residuosity (modulo composite integers whose factorization is not known) is computationally hard, it is shown that the NP-complete language of satisfiability also possesses noninteractive zero-knowledge proofs.

Key words. interactive proofs, randomization, zero-knowledge proofs, secure protocols, cryptography, quadratic residuosity

AMS(MOS) subject classifications. 68Q15, 94A60

1. Introduction. *Zero-knowledge proofs.* Recently, Goldwasser, Micali, and Rackoff [GoMiRa] have shown that it is possible to prove that some theorems are true without giving the slightest hint of why this is so. This is rigorously formalized in the somewhat paradoxical notion of a *zero-knowledge proof system* (ZKPS).

Zero-knowledge proofs have proven to be very useful both in Complexity Theory and in Cryptography. For instance, in Complexity Theory, via results of Fortnow [Fo] and Boppana, Hastad, and Zachos [BoHaZa], zero-knowledge provides us an avenue to convince ourselves that certain languages are not NP-complete. In cryptography, zero-knowledge proofs have played a major role in the recently proven completeness theorem for protocols with honest majority [GoMiWi2], [ChCrDa], and [BeGoWi]. They also have inspired rigorously analyzed identification schemes [FeFiSh], [MiSh] that are as efficient as folklore ones.

The ingredients of zero-knowledge. Despite its wide applicability, zero-knowledge remains an intriguing notion: What makes zero-knowledge proofs work?

Three main ingredients differentiate standard zero-knowledge proofs from more traditional ones:

1. *Interaction*: The prover and the verifier talk back and forth.
2. *Hidden Randomization*: The verifier tosses coins that are hidden from the prover and thus unpredictable to him.
3. *Computational Difficulty*: The prover embeds in his proofs the computational difficulty of some other problem.

In sum, quite a rich scenario is needed for implementing zero-knowledge proofs. Can one achieve the same results "with fewer ingredients"? Properly answering this question is the goal of this paper. Any such answer is not only important from a purely

* Received by the editors September 4, 1990; accepted for publication February 15, 1991.

† Computer Science Department, University of California, Berkeley, California 94720. This author's research was supported by National Science Foundation grant # DCR85-13926.

‡ IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York 10598. The work of this author was done at IBM, while he was on leave from Dipartimento di Informatica ed Applicazioni, Università di Salerno, Salerno, Italy.

§ Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The works of this author was supported by National Science Foundation grant # CCR-8719689.

¶ Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138. The work of this author was partially supported by Office of Naval Research grant # N00039-88-C-0163.

theoretical point of view, but from a practical one as well: the ability to implement zero-knowledge proofs in "poorer" settings would greatly enhance the applicability of these ideas.

1.1. A new, simpler scenario for zero-knowledge. *The new goal.* Let A and B be two mathematicians. A leaves for a long trip around the world, during which he continues his mathematical investigations. We want to enable him, whenever he discovers the proof of a new theorem, to write a postcard to B proving the validity of his assertion in zero-knowledge. This is a noninteractive process. Better, it is a monodirectional interaction: from A to B only. In fact, even if B would like to answer, he couldn't: A has no stable (or predictable) address and will move away before any mail can reach him.

The new scenario. Achieving the new goal is a bit tricky. Without any shared information, "monodirectional" and zero-knowledge proofs are possible only for trivial statements. We shall see, however, that, under a complexity assumption, such proofs exist for any "NP theorem" thanks to a simple, innocent-looking, ingredient: shared randomness. That is, both prover and verifier have access to the same, short, random string.

Past and present. Blum, Feldman, and Micali [BlFeMi] were the first to conceive that zero-knowledge proofs could be based on the above, simple ingredient, and proposed the name of noninteractive zero-knowledge proofs for them, and presented some noninteractive zero-knowledge proofs. De Santis, Micali, and Persiano [DeMiPe1] improved on their results by using a weaker complexity assumption. The present paper summarizes and improves on both these results.

First, we contribute a crisper formalization of noninteractive zero-knowledge; second, we modify their algorithms and provide a full proof of correctness for them, thus removing a subtle bug (pointed out by Bellare) in some part of their argument.[1]

1.2. Shared random strings and public coins. As we have said, we have prover and verifier share a common, random string. Actually, in our proof systems the verifier will not toss any secret coins at all.

The idea of protocols with public randomness is not new. Protocols making use of public randomness were already known in the literature, both in a cryptographic and in a complexity-theoretic scenario. These protocols, however, were developed for quite different ends, and differ from our scenario in the way the coin tosses are made available.

Random beacons. In [Ra3], Rabin presents the notion of a random beacon. This is a source broadcasting random bits at regular time intervals. He used this device for "achieving simultaneity" in contract signing.

Note, though, that sharing a common random string is a requirement *weaker* than having both parties access a random beacon (e.g., sharing the same Geiger counter). In this latter case, in fact, all made coin tosses would be seen by both parties, but the future ones would still be unpredictable. By contrast, our model allows the prover to see in advance the outcome of all the coin tosses the verifier will ever make. That is, the zero-knowledgeness of our proofs does not depend on the secrecy or unpredictability of σ but on the "well mixedness" of its bits![2]

---

[1] The part that presented a problem in their argument was the one relative to "many-theorems," that is, the equivalent of our §6.

[2] This curious property makes our result potentially applicable. For instance, all libraries in the country possess identical copies of the random tables prepared by the Rand Corporation. Thus, we may think of ourselves as being already in the scenario needed for noninteractive zero-knowledge

Note that sharing a random string σ is a weaker requirement than being able to interact. In fact, if A and B could interact, they would be able to construct a common random string, for instance, by coin tossing over the phone [Bl1]; the converse, however, is not true.

Arthur–Merlin games. The question of the power of hidden randomness versus public randomness has already been discussed in Complexity Theory in the context of proof systems. Goldwasser, Micali, and Rackoff [GoMiRa] and Babai and Moran [Ba], [BaMo] consider proofs as games played between two players, prover and verifier, who can talk back and forth. In [GoMiRa], the verifier is allowed to flip fair coins and hide their outcomes from the prover. In [Ba], [BaMo], all coin tosses made by the verifier are seen by the prover—called, respectively, Arthur and Merlin in proof systems of this type. Actually, each message from the verifier to the prover consists of a random string. Thus in an Arthur–Merlin proof system, the verifier can be substituted by a random beacon: rather than having the verifier send his next message, one waits for the next transmission of the beacon. That is, once again, all made coin tosses are publicly known, but future ones are still unpredictable. Only if the verifier is guaranteed to send a single message are we in a shared-random-string scenario. The class of languages recognized by such a restricted proof system is denoted by "AM_2" or "$AM[2]$" (to specify that there are exactly two rounds of communication). We show that, under proper complexity assumptions, this class coincides with the set of languages possessing noninteractive zero-knowledge proofs.

1.3. Applications of noninteractive zero-knowledge. Powerful computer networks are in place, and can be used for executing a huge variety of cryptographic protocols. Zero-knowledge proofs are crucial to these protocols and, at the same time, interaction is the most expensive resource.[3] Thus noninteractive zero-knowledge proofs may be used to save precious communication rounds in cryptographic protocols.

Besides this, noninteractive zero-knowledge has been used by Bellare and Goldwasser [BeGo] as an alternative basis for secure digital signatures (in the sense of [GoMiRi]). Also, following a hint of [BlFeMi], Naor and Yung [NaYu] exhibit public-key cryptosystems secure against chosen cipher-text attack.

1.4. Organization. The next section is devoted to setting up our notation, recalling some elementary facts from Number Theory, and stating the complexity assumption which suffices to show the existence of noninteractive ZKPS.

In §3 we define the notion of bounded noninteractive zero-knowledge; that is, the "single theorem" case.

In §4 we show that a special number-theoretic language L possesses a bounded noninteractive zero-knowledge proof. That is, if prover and verifier share a random string, then it is possible to prove, noninteractively and in zero-knowledge, that any single, sufficiently shorter $x \in L$.

In §5, under the quadratic residuosity assumption, we prove that the "more general" language of $3SAT$ is in bounded noninteractive zero-knowledge.

Only in §6 do we show that, if deciding quadratic residuosity is hard, the prover can show in zero-knowledge membership in NP languages for any number of strings, each of arbitrary size, using the *same* randomly chosen string.

In §7 we will discuss some related work.

proofs.

[3] The internal computation of a typical cryptographic protocol can be performed in a few seconds, but the time it takes to exchange electronic mail a hundred times may not be negligible.

In §8 we will state an open problem that we would love to see solved.

2. Preliminaries.

2.1. Basic definitions. *Notation.* We denote by $\mathcal{N}$ the set of natural numbers. If $n \in \mathcal{N}$, by 1^n we denote the concatenation of n 1's. We identify a binary string σ with the integer x whose binary representation (with possible leading zeros) is σ.

By the expression $|x|$ we denote the length of x if x is a string, the length of the binary string representing x if x is an integer, the absolute value of x if x is a real number, or the cardinality of x if x is a set.

If σ and τ are binary strings, we denote their concatenation by either $\sigma \circ \tau$ or $\sigma\tau$.

A language is a subset of $\{0,1\}^*$. If L is a language and $k > 0$, we set $L_k = \{x \in L \colon |x| \leq k\}$. For variety of discourse, we may call "theorem" a string belonging to the language at hand. (A "false theorem" is a string outside L.)

Models of computation. An algorithm is a Turing machine. An *efficient* algorithm is a probabilistic Turing machine running in expected polynomial time.

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 inputs, we write "$A(\cdot,\cdot)$" and so on.

A sequence of probabilistic Turing machines $\{T_n\}_{n \in \mathcal{N}}$ is an *efficient nonuniform algorithm* if there exists a positive constant c such that, for all sufficiently large n, T_n halts in expected n^c steps and the size of its program is less than or equal to n^c. We use efficient nonuniform algorithms to gain the power of using different Turing machines for different input lengths. For instance, T_n can be used for inputs of length n. The power of nonuniformity lies in the fact that each Turing machine in the sequence may have "wired-in" (i.e., properly encoded in its program) a small amount of special information about its own input length.[4]

A *random selector* is a special (random) oracle. The oracle query consists of a pair of strings $(s, \mathcal{S})$, where the second string encodes a finite set. Such a query is answered by the oracle with a randomly chosen element in the set $\mathcal{S}$. If the oracle is asked the same query twice, it will return the same element. The role of the first entry in the query is to allow us, if so wanted, to make random an independent selection in a set $\mathcal{S}$. That is, if $\mathcal{S}$ is the same, and $s_1 \neq s_2$, then, in response to queries $(s_1, \mathcal{S})$ and $(s_2, \mathcal{S})$, the oracle will return two elements from $\mathcal{S}$, each randomly and independently selected.

A *random selecting algorithm* is a Turing machine with access to a random selector. Note that a random selecting algorithm is strictly more powerful than one with access to coin or random oracle. For instance, a random selecting algorithm can select with uniform probability one out of three elements. On the other hand, simulating independent coin flips is easy with a random selector: If *Select* is a random selector, to ensure the independence of b_i, the ith coin flip, from all the other coin flips in a computation on input x, one can set $b_i = \mathit{Select}(x \circ i, \{0,1\})$.

Random selectors will simplify the description of our algorithms. In fact, we desire a prover in a noninteractive proof system to be "memoryless." That is, it needs not remember which theorems it proved in the past to find and prove the next theorem. However, for zero-knowledge purposes, it will be much handier to keep track of some history, the history, that is, of previously made coin tosses. This will be crucial in §6. A random selector will, in fact, accomplish this record-keeping without having

[4] This definition can be shown to be equivalent to the one of a poly-size combinatorial circuit and to the one [KaLi] of poly-time Turing machine that takes advice.

to consider provers "with history." As we shall point out, random selectors can be efficiently approximated, and thus only represent a conceptual tool.

Algorithms and probability spaces. If $A(\cdot)$ is a probabilistic algorithm, then for any input x, the notation $A(x)$ refers to the probability space that assigns to the string σ the probability that A, on input x, outputs σ.

Following the notation of [GoMiRi], if S is a probability space, then "$x \stackrel{R}{\leftarrow} S$" denotes the algorithm which assigns to x an element randomly selected according to S. If F is a finite set, then the notation "$x \stackrel{R}{\leftarrow} F$" denotes the algorithm which assigns to x an element selected according to the probability space whose sample space is F and uniform probability distribution on the sample points.

If $p(\cdot,\cdot,\cdots)$ is a predicate, the notation $Pr(x \stackrel{R}{\leftarrow} S; y \stackrel{R}{\leftarrow} T; \cdots : p(x,y,\cdots))$ denotes the probability that $p(x,y,\cdots)$ will be true after the ordered execution of the algorithms $x \stackrel{R}{\leftarrow} S$, $y \stackrel{R}{\leftarrow} T, \cdots$.

The notation $\{x \stackrel{R}{\leftarrow} S; y \stackrel{R}{\leftarrow} T; \cdots : (x,y,\cdots)\}$ denotes the probability space over $\{(x,y,\cdots)\}$ generated by the ordered execution of the algorithms $x \stackrel{R}{\leftarrow} S$, $y \stackrel{R}{\leftarrow} T, \cdots$.

2.2. Number theory. *Quadratic Residuosity.* For each integer $x > 0$, the set of integers less than x and relatively prime to x form a group under multiplication modulo x denoted by Z_x^*. We say that $y \in Z_x^*$ is a *quadratic residue* modulo x if and only if there is a $w \in Z_x^*$ such that $w^2 \equiv y \bmod x$. If this is not the case, we call y a *quadratic nonresidue* modulo x. For compactness, we define the *quadratic residuosity predicate* as follows:

$$\mathcal{Q}_x(y) = \begin{cases} 0 & \text{if } y \text{ is a quadratic residue modulo } x, \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

FACT 2.1 (see for instance [NiZu]). If $y_1, y_2 \in Z_x^*$, then

1. $\mathcal{Q}_x(y_1) = \mathcal{Q}_x(y_2) = 0 \Longrightarrow \mathcal{Q}_x(y_1y_2) = 0$.
2. $\mathcal{Q}_x(y_1) \neq \mathcal{Q}_x(y_2) \Longrightarrow \mathcal{Q}_x(y_1y_2) = 1$.

The quadratic residuosity predicate defines the following equivalence relation in Z_x^*: $y_1 \sim_x y_2$ if and only if $\mathcal{Q}_x(y_1y_2) = 0$. Thus, the quadratic residues modulo x form a $\sim_x$ equivalence class. More generally, Fact 2.2 is immediately seen.

FACT 2.2. For any fixed $y \in Z_x^*$, the elements $\{yq \bmod x \mid q \text{ is a quadratic residue modulo } x\}$ constitute a $\sim_x$ equivalence class that has the same cardinality as the class of quadratic residues.

The problem of deciding quadratic residuosity consists of evaluating the predicate $\mathcal{Q}_x$. As we now see, this is easy when the modulus x is prime and appears to be hard when it is composite.

Prime moduli. Primes are easy to recognize.

FACT 2.3 ([AdHu] extending [GoKi]). There exists an efficient algorithm that, on input x, outputs YES if and only if x is prime.

For p prime, the problem of deciding quadratic residuosity coincides with the problem of computing the Legendre symbol. In fact, for p prime and $y \in Z_p^*$, the Legendre symbol $(y|p)$ of y modulo p is defined as

$$(y|p) = \begin{cases} +1 & \text{if } y \text{ is a quadratic residue modulo } x, \text{ and} \\ -1 & \text{otherwise;} \end{cases}$$

and can be computed in polynomial time by using Euler's criterion. Namely,

$$(y|p) = y^{(p-1)/2} \bmod p.$$

Composites are easy to recognize. It is easy to test compositeness.

FACT 2.4 ([Ra1], [SoSt]). There exists a polynomial-time algorithm $TEST(\cdot, \cdot)$ such that

1. if x is composite, $TEST(x, r)$ =COMPOSITE for at least $\frac{3}{8}$ of the strings r such that $|r| = |x|$.
2. if x is prime, $TEST(x, r)$ =PRIME for all r's.

We say that the sequence $(p_1, h_1), \cdots, (p_n, h_n)$ is the *factorization* of x if the p_i's are distinct primes, the h_i's are positive integers, and $x = \prod_{i=1}^{n} p_i^{h_i}$.

While it is easy to test compositeness, no efficient algorithm is known for computing the factorization of a composite integer. In fact, the following assumption is consistent with our state of knowledge.

Factoring assumption. For each efficient nonuniform algorithm $C = \{C_n\}_{n \in \mathcal{N}}$, let p_n^C denote the probability that, on inputing an integer x product of two randomly selected primes of length n, C_n outputs—in some standard encoding—the factorization of x. (This probability is computed over all possible choices of the two primes and the internal coin tosses of C_n.) Then for all positive constants d, and all sufficiently large n, $p_n^C < n^{-d}$.

Often, computational problems relative to composite moduli are easy if their factorization is known. For example, this is the case for the problem of computing square roots modulo x.

FACT 2.5 (see for instance [An]). There exists an efficient algorithm that, given as inputs x, its prime factorization, and y, a quadratic residue modulo x, outputs a random square root of y modulo x.

FACT 2.6 ([Ra2]). The problem of factoring composite integers is probabilistic polynomial-time reducible to the problem of extracting square roots modulo composite integers.

Another computational problem modulo x that is easy given the factorization of x is deciding quadratic residuosity.

FACT 2.7 (see, for instance, [NiZu]). y is a quadratic residue modulo x if and only if y is a quadratic residue modulo each of the prime divisors of x.

However, no efficient algorithm is known for deciding quadratic residuosity modulo composite numbers whose factorization is not given. Some help is provided by the Jacobi symbol, which extends the Legendre symbol to composite integers as follows. Let $(p_1, h_1), \cdots, (p_n, h_n)$ be the prime factorization of x, and $y \in Z_x^*$. Then[5]

$$(y|x) = \prod_{i=1}^{n} (y|p_i)^{h_i}.$$

Define J_x^{+1} and J_x^{-1} to be, respectively, the subsets of Z_x^* whose Jacobi symbol is $+1$ and -1. It can be immediately seen that if $y \in J_x^{-1}$, then it is not a quadratic residue modulo x, as it is not a quadratic residue modulo some prime p_i dividing x. However, if $y \in J_x^{+1}$, no efficient algorithm is known to compute $\mathcal{Q}_x(y)$. Actually, the fastest way known consists of first factoring x and then computing $\mathcal{Q}_x(y)$. This fact was first used in cryptography by Goldwasser and Micali [GoMi1]. We will use it in this paper with respect to the following special moduli.

[5] Despite the fact that the Jacobi symbol is defined in terms of the factorization of the modulus, it can be computed in polynomial time. (This can be derived by a time analysis of the classical algorithm presented in [NiZu]; see also [An].)

Blum integers. For $n \in \mathcal{N}$, we define the set of Blum integers of size n, $BL(n)$, as follows: $x \in BL(n)$ if and only if $x = pq$, where p and q are primes of length n, both congruent to $3 \bmod 4$. These integers were first used for cryptographic purposes by [Bl1].

Blum integers are easy to generate. By Fact 2.3 and the density of the primes congruent to $3 \bmod 4$ (de la Vallee Poussin's extension of the prime number theorem [Sh]), it is easy to prove the following.

FACT 2.8. There exists an efficient algorithm that, on input 1^n, outputs the factorization of a randomly selected $x \in BL(n)$.

This class of integers constitutes the hardest input for any known efficient factoring algorithm. Thus no efficient algorithm is known for deciding quadratic residuosity modulo Blum integers, which justifies the following.

Quadratic Residuosity Assumption (QRA). For each efficient nonuniform algorithm $\{C_n\}_{n\in\mathcal{N}}$, all positive constants d, and all sufficiently large n,

$$Pr\Big(x \stackrel{R}{\leftarrow} BL(n);\ y \stackrel{R}{\leftarrow} J_x^{+1} : C_n(x,y) = \mathcal{Q}_x(y)\Big) < \frac{1}{2} + n^{-d}.$$

That is, no efficient nonuniform algorithm can guess the value of the quadratic residuosity predicate substantially better than by random guessing.

It follows from Fact 2.7 and Euler's criterion that, if x is a Blum integer, $-1 \bmod x$ is a quadratic nonresidue with Jacobi symbol $+1$.

FACT 2.9. On input of a Blum integer x, it is easy to generate a random quadratic nonresidue in J_x^{+1}: randomly select $r \in Z_x^*$ and output $-r^2 \bmod x$.

Regular integers. A Blum integer enjoys an elegant structural property. Namely, $|J_x^{+1}| = |J_x^{-1}|$. More generally, we define an integer x to be *regular* if it enjoys the above property. We define *Regular*(s) to be the set of regular integers with s distinct prime divisors. By the definition of Jacobi symbol, Fact 2.10 is straightforward.

FACT 2.10. An odd integer x belongs to *Regular*(s) if and only if it has s distinct prime factors and is not a perfect square.

Equivalently, by Fact 2.2, we have Fact 2.11.

FACT 2.11. An odd integer x belongs to *Regular*(s) if and only if it is regular and Z_x^* is partitioned by $\sim_x$ into 2^s equally numerous equivalence classes. (Equivalently, J_x^{+1} is partitioned by $\sim_x$ into 2^{s-1} equally numerous equivalence classes.)

3. Bounded noninteractive zero-knowledge proofs. A bounded noninteractive zero-knowledge proof system is a special algorithm. Given as input a random string σ and a single, sufficiently shorter theorem T, it outputs a second string that will convince (noninteractively and) in zero-knowledge that T is true for any verifier who has access to the same σ. It is important in this process that a "brand new" random string is employed for each theorem. The word "bounded" refers to the fact that if the same σ is used over and over again for convincing the verifier of the validity of many theorems, the produced noninteractive proofs may no longer be zero-knowledge.

DEFINITION 3.1. Let A_1 and A_2 be Turing machines. We say that (A_1, A_2) is a *sender–receiver* pair if their computation on a *common input* x works as follows. First, algorithm A_1, on input x, outputs a string m_x. Then, algorithm A_2 computes on inputs x and m_x and outputs ACCEPT or REJECT. If (A_1, A_2) is a sender–receiver pair, A_1 is called the sender and A_2 the receiver. The running time of both machines is calculated only in terms of the common input.

Thus m_x can be interpreted as a message sent by A_1 to A_2.

Notation. In our sender–receiver pairs, the output of the sender is described in terms of s "send instructions," where s depends solely on the input length. If "send v" is the ith such instruction, this is shorthand for "output (i, v)." Without explicitly saying it, the receiver always checks that for each $i = 1, \cdots, s$, exactly one pair with first entry i is received. If this is not the case, or if the second component of a pair is not of the right form (i.e., is not of the proper length, is a string rather than a set, etc.), the receiver immediately halts outputting REJECT. Thus if "send v" is the ith instruction of the sender, "check that $v \cdots$" means "check that the second component of the pair whose first entry is $i \cdots$." That is, the receiver parses without ambiguity the sender's output.

DEFINITION 3.2. Let (*Prover, Verifier*) be a sender–receiver pair where $Prover(\cdot,\cdot)$ is random selecting and $Verifier(\cdot,\cdot,\cdot)$ is polynomial time. We say that (*Prover, Verifier*) is a bounded noninteractive zero-knowledge proof system (bounded noninteractive ZKPS) for the language L if there exists a positive constant c such that:

1. *Completeness.* For all $x \in L_n$ and for all sufficiently large n,

$$Pr(\sigma \xleftarrow{R} \{0,1\}^{n^c}; Proof \xleftarrow{R} Prover(\sigma, x) \colon Verifier(\sigma, x, Proof) = 1) > \frac{2}{3}.$$

2. *Soundness.* For all $x \notin L_n$, for all Turing machines $Prover'$, and for all sufficiently large n,

$$Pr(\sigma \xleftarrow{R} \{0,1\}^{n^c}; Proof \xleftarrow{R} Prover'(\sigma, x) \colon Verifier(\sigma, x, Proof) = 1) < \frac{1}{3}.$$

3. *Zero-Knowledge.* There exists an efficient algorithm S such that for all $x \in L_n$, for all efficient nonuniform (distinguishing) algorithms D, for all $d > 0$, and, all sufficiently large n,

$$\left| Pr(s \xleftarrow{R} View(n, x) : D_n(s) = 1) - Pr(s \xleftarrow{R} S(1^n, x) : D_n(s) = 1) \right| < n^{-d},$$

where

$$View(n, x) = \left\{ \sigma \xleftarrow{R} \{0,1\}^{n^c}; Proof \xleftarrow{R} Prover(\sigma, x) \colon (x, \sigma, Proof) \right\}.$$

We call *Simulator* the algorithm S.

We define the class of languages *Bounded*-NIZK as follows:

$$\textit{Bounded}\text{-NIZK} = \{L \colon L \text{ has a bounded noninteractive ZKPS}\}.$$

A sender–receiver pair (*Prover, Verifier*) is a bounded noninteractive proof system for the language L if there exists a positive constant c such that completeness and soundness hold (such a c will be referred to as the *constant* of (*Prover, Verifier*)). We let *bounded noninteractive P* be the class of languages L having a bounded noninteractive proof system.

We call the "common" random string σ, input to both *Prover* and *Verifier*, the *reference string.* (Above, the common input is σ and x.)

Discussion. *Proving and verifying.* As usual, we do not care how difficult it is to prove a true theorem, but we do insist that verifying is always easy. Thus, we have chosen our prover as powerful as possible, though it cannot use its power to find "long" proofs, since the verifier is polynomial time (in the common input).

Arthur–Merlin games. It is immediately seen that the notion of a bounded noninteractive proof system is equivalent to that of a two-move Arthur–Merlin Proof System [Ba], [BaMo]. Thus, letting AM_2 denote the class of languages accepted by a two-move Arthur–Merlin Proof System, we have *Bounded*-NIZK $\subseteq AM_2$. Actually, as we shall prove in §5.5, this containment is an equality under a proper complexity assumption.

Deterministic verification. Note that our verifiers are defined to be deterministic. Thus, if they want to perform some probabilistic computation, they are forced to use part of the reference string. A cheating prover may thus try to exploit this fact to his advantage.

Probability enhancement. As for the case of BPP algorithms and interactive proofs, the definition of completeness and soundness is independent of the constants $\frac{2}{3}$ and $\frac{1}{3}$. In fact, these (or other "bounded away") probabilities can be pumped up (and down) easily by repeating the proving process sufficiently many times, each using a distinct segment of a sufficiently longer reference string. This process is called "parallel composition." However, as noted by Micali, for the case of interactive zero-knowledge proofs, parallel composition may also enhance the amount of knowledge released! Indeed, zero-knowledge proofs do not appear to be closed under parallel composition. The reason for which straightforward parallel composition fails in the case of interactive zero-knowledge proofs is precisely that the *interaction* may be exploited in subtle ways by a "cheating verifier."[6] One advantage of noninteractive zero-knowledge is exactly the fact that one does not have to worry about "cheating" verifiers: as is immediately seen, bounded noninteractive zero-knowledge is closed under parallel composition.

Completeness means that (after a sufficient enhancement) the probability of succeeding in proving a true theorem T is overwhelming. This is so even if T is selected after the string σ has been chosen. More precisely, a simple counting argument shows that completeness is equivalent to the following:

1′. *Strong completeness.* For all probabilistic algorithms *Choose-in-L*$(\cdot)$ that, on inputting an n^c-bit string, return elements in L_n, and all sufficiently large n,

$$Pr(\sigma \stackrel{R}{\leftarrow} \{0,1\}^{n^c}; \quad x \stackrel{R}{\leftarrow} \mathit{Choose\text{-}in\text{-}L}(\sigma); \quad \mathit{Proof} \stackrel{R}{\leftarrow} \mathit{Prover}(\sigma, x) : \mathit{Verifier}(\sigma, x, \mathit{Proof}) = 1) > 1 - 2^{-n}.$$

That strong completeness holds can be seen by first using parallel composition so as to replace the probability $\frac{2}{3}$ of completeness with $1 - 2^{-2n}$, and then noticing that there are at most 2^n theorems of length n.

Actually, completeness can be replaced by an even simpler property. Namely,

1″. *Perfect completeness.* For all $x \in L_n$,

$$Pr(\sigma \stackrel{R}{\leftarrow} \{0,1\}^{n^c}; \mathit{Proof} \stackrel{R}{\leftarrow} \mathit{Prover}(\sigma, x) : \mathit{Verifier}(\sigma, x, \mathit{Proof}) = 1) = 1.$$

In fact, we have Theorem 3.3.

THEOREM 3.3. *Let $L \in$ Bounded-NIZK. Then L has a bounded noninteractive ZKPS with perfect completeness.*

Proof. Furer et al. [FuGoMaSiZa] have proved that any AM_2 language has an interactive proof system with perfect completeness. Now let (P, V) be a bounded

[6] Elaborating on this subtle point is not within the scope of this paper. For an explanation of it (and pointers to related results) see [BeMiOs].

noninteractive ZKPS for L for which completeness holds with overwhelming probability. Then modify P as follows. Whenever the proof generated by P is not accepted by the verifier (something that can be easily computed), as bounded noninteractive P$=AM_2$, the new prover interprets the reference string as an Arthur move, and responds with a Merlin move so as to achieve perfect completeness. This extra step guarantees that the verifier will always be convinced (of a true theorem), and thus perfect completeness holds. It is immediately seen that soundness keeps on holding. Also, zero-knowledge keeps on holding: the extra step may be "dangerous," but it is performed only too rarely.

Soundness means that the probability of succeeding in proving a false theorem T is negligible. This still holds if T is chosen after σ has been selected. More precisely, a simple counting argument shows that soundness is equivalent to

2$'$. *Strong soundness.* For all probabilistic algorithms *Adversary* outputting pairs $(x, Proof)$, where $x \notin L_n$, and all sufficiently large n,

$$Pr(\sigma \xleftarrow{R} \{0,1\}^{n^c}; (x, Proof) \xleftarrow{R} \mathit{Adversary}(\sigma):\ \mathit{Verifier}(\sigma, x, Proof) = 1) < 2^{-n}.$$

Zero-knowledge guarantees that the proof gives no knowledge but the validity of the theorem. All the verifier may see in our scenario, σ and *Proof*, can be efficiently computed with essentially the same odds without "knowing how to prove T."

Note that in our scenario, the definition of zero-knowledge is simpler than the one in [GoMiRa]. As there is no interaction between verifier and prover, we do not have to worry about possible cheating by the verifier to obtain a "more interesting view." That is, we can eliminate the quantification "$\forall\, \mathit{Verifier}'$" from the original definition of [GoMiRa].

Analogously to [GoMiRa], we may define a bounded noninteractive proof system (*Prover, Verifier*) to be *perfect zero-knowledge* if the following more stringent condition holds:

3$'$. *Perfect zero-knowledge.* There exists an efficient algorithm S such that for all $x \in L_n$ and all sufficiently large n,

$$View(n, x) = S(1^n, x),$$

where

$$View(n, x) = \left\{\sigma \xleftarrow{R} \{0,1\}^{n^c}; \mathit{Proof} \xleftarrow{R} \mathit{Prover}(\sigma, x):\ (\sigma, \mathit{Proof})\right\}.$$

Thus the notion of perfect ZK is independent of the computing power of "the observer/distinguisher."

While for completeness and soundness it is not important whether the true/false theorem is chosen before or after the reference string, this need not to be the case for zero-knowledge. It is actually important that the prover chooses the true theorem T he wants to prove *independently* of σ. This, in practice, is not a restriction, since σ does not have any special meaning. The sole purpose of σ is to provide a common source of randomness, and thus it can be accessed only *after* the prover has chosen which theorem to prove, in which case the "independence" condition is automatically satisfied. Should the prover want to prove a statement "about" the reference string, there is no guarantee that no knowledge would be revealed, while there is still a guarantee that the statement cannot be false.

4. A bounded noninteractive ZKPS for a special language.

DEFINITION 4.1. Set $\mathcal{QR} = \cup_n \mathcal{QR}(n)$ and $\mathcal{NQR} = \cup_n \mathcal{NQR}(n)$, where

$$\mathcal{QR}(n) = \{(x, y) \mid x \in \mathit{Regular}(2),\ |x| \leq n,\ \text{ and } \mathcal{Q}_x(y) = 0\}$$

and

$$\mathcal{NQR}(n) = \{(x, y) \mid x \in \mathit{Regular}(2),\ |x| \leq n,\ y \in J_x^{+1},\ \text{ and } \mathcal{Q}_x(y) = 1\}.$$

If one restricts the modulo x in the definition of $\mathcal{QR}$ and $\mathcal{NQR}$ to be a Blum integer, then the quadratic residuosity assumption states that it is hard to distinguish the languages $\mathcal{QR}$ and $\mathcal{NQR}$.

For $x \in \mathit{Regular}(2)$, QR_x denotes the set $\{y \mid (x, y) \in \mathcal{QR}\}$ and NQR_x the set $\{y \mid (x, y) \in \mathcal{NQR}\}$.

DEFINITION 4.2. If $(x, y) \in \mathcal{NQR}$ and $z \in J_x^{+1}$, we say that $s \in Z_x^*$ is an (x, y)-*root* of z if $z = s^2 \bmod x$ or $zy = s^2 \bmod x$. (Note that only one of the two cases may apply.) If s is an (x, y)-root of z, we write $s = \sqrt[(x,y)]{z}$.

In this section we prove that $\mathcal{NQR}$ has a bounded noninteractive proof system that is perfect zero-knowledge. The proof system below is based on an earlier protocol of Goldwasser and Micali [GoMi2].

The Sender–Receiver Pair (A,B)
Input to A and B:
- $(x, y) \in \mathcal{NQR}(n)$
- A n^3-bit random string ρ.
  (Set $\rho = \rho_1 \rho_2 \cdots \rho_{n^2}$, where each ρ_i has length n.)

Instructions for A:
- For $i = 1, \cdots, n^2$, if $\rho_i \in J_x^{+1}$, then randomly choose and send $s_i = \sqrt[(x,y)]{\rho_i}$.

Instructions for B:

B.0. If $\rho_i \in J_x^{+1}$ for less than $3n$ of the indices i, then stop and ACCEPT. Else,
B.1. Verify that x is odd and that $y \in J_x^{+1}$. If not, stop and REJECT. Else,
B.2. Verify that x is not a perfect square. If not, stop and REJECT. Else,
B.3. If x is a prime power, stop and REJECT. Else,
B.4. For each $\rho_i \in J_x^{+1}$ verify that $s_i = \sqrt[(x,y)]{\rho_i}$. If not, stop and REJECT. Else ACCEPT.

THEOREM 4.3. *(A, B) is a bounded noninteractive* ZKPS *for $\mathcal{NQR}$.*

Proof. First, (A, B) is a sender–receiver pair. Second, B runs in polynomial time. In fact, the Jacobi symbol can be computed in polynomial time, steps B.2 and B.4 are trivial, and step B.3 can be performed as follows:

B.3.1. Compute the largest integer α for which $x = w^\alpha$ for some $w \in \mathcal{N}$. (Only values $1, \cdots, |x|$ should be tried for α and a binary search can be performed for finding w, if it exists.)
B.3.2. Compute z such that $z^\alpha = x$.
B.3.3. If for all $1 \leq i \leq n^2$, $TEST(z, \rho_i)$ =PRIME, stop and REJECT.

Third, properties 1–3 of a bounded noninteractive ZKPS also hold.

Completeness. We actually prove that strong completeness holds. This implies that the weaker property 1 also holds. If $(x, y) \in \mathcal{NQR}(n)$, then step B.1 is trivially passed. Step B.2 is passed because of Fact 2.10. B.3 is passed with probability greater than $1 - 2^{-n}$. This can be argued as follows. For any fixed $\overline{x} \in \mathit{Regular}(2)$, the probability that $TEST$ outputs PRIME on a single ρ_i is at most $\frac{5}{8}$, and thus (since the ρ_i's are independent) the probability that B.3 is not successfully passed is at most $(\frac{5}{8})^{n^2}$. Since there are at most 2^n x's such that $(x, z) \in \mathcal{NQR}(n)$ for some z, the probability that step B.3 is not successfully passed is at most $2^n(\frac{5}{8})^{n^2} \leq 2^{-n}$. Finally, step B.4 is passed with probability 1. In fact, as $x \in \mathit{Regular}(2)$, by Fact 2.11, there are exactly 2 $\sim_x$ equivalence classes in J_x^{+1}. That is, either ρ_i is a quadratic residue modulo x or ρ_i is in the same equivalence class as y, in which case $y\rho_i$ is a quadratic residue.

Soundness. As for the completeness property, we actually prove that strong soundness holds.

First, observe that B stops at step B.0 only with negligible probability. Indeed, for a fixed $\overline{x}$, the probability that $\rho_i \in J_{\overline{x}}^{+1}$ is greater than $\frac{1}{8}$. By the Chernoff bound (see [AnVa] and [ErSp]), the probability that $\rho_i \in J_{\overline{x}}^{+1}$ for fewer than $3n$ of the indices is (for large n) less than 2^{-2n}. Thus, the probability that there is an x for which B stops at step B.0 is at most $2^n 2^{-2n} = 2^{-n}$.

Assume that $(x, y) \notin \mathcal{NQR}$. Then, either (a) $x \in \mathit{Regular}(2)$ but $\mathcal{Q}_x(y) = 0$, or (b) $x \notin \mathit{Regular}(2)$. For any fixed input $(\overline{x}, \overline{y})$ for which case (a) occurs, the probability that B.4 is successfully passed is at most 2^{-3n}. (In fact, B.4 is passed if and only if all ρ_i's are quadratic residues modulo x.) Thus, the probability that step B.4 is passed, for any input for which case (a) occurs, is at most $2^n 2^{-3n} = 2^{-2n}$.

Consider now the case that $(x, y) \notin \mathcal{NQR}$ because of reason (b). Then either (b.1) x is not regular, or (b.2) $x \in \mathit{Regular}(1)$, or (b.3) $x \in \mathit{Regular}(s)$ for $s \geq 3$. In case (b.1), due to Fact 2.10, an odd x must be a perfect square which would be detected in step B.2. In case (b.2), x is a prime power which would be detected by step B.3. Let us now argue case (b.3). For any fixed $(\overline{x}, \overline{y})$ with $\overline{x} \in \mathit{Regular}(s)$, $s \geq 3$, the probability that step B.4 is successfully passed is at most 2^{-n}. (In fact, this would happen only if, for each $\rho_i \in J_{\overline{x}}^{+1}$, either ρ_i or $\rho_i y$ is a quadratic residue modulo $\overline{x}$. This happens with probability smaller than or equal to $\frac{1}{2}$ since, because of Fact 2.11, there are at least four $\sim_x$ equivalence classes in $J_{\overline{x}}^{+1}$.) Thus the probability that, for any input outside $\mathcal{NQR}$ because of reason (b.3), step B.4 is successfully passed is at most $2^{2n} 2^{-3n} = 2^{-n}$.

Zero-knowledge. Let us specify a (simulating) efficient algorithm M that, on input $(x, y) \in \mathcal{NQR}$, generates a random variable which no algorithm can distinguish from B's view on input $(x, y) \in \mathcal{NQR}$.

M's program
Input: $(x, y) \in \mathcal{NQR}(n)$.
1. Set $Proof$ = empty string.
2. For $i = 1$ to n^2
   - Randomly select an n-bit integer s_i, with possible leading 0's.
   - If $s_i \notin J_x^{+1}$ then set $\rho_i = s_i$.
     - else
       - Toss a fair coin.
       - If HEAD set $\rho_i = s_i^2 \bmod x$ and append s_i to $Proof$.
       - If TAIL set $\rho_i = y^{-1} s_i^2 \bmod x$ and append s_i to $Proof$.
3. Set $\rho = \rho_1 \cdots \rho_{n^2}$.

Output: $(\rho, Proof)$.

Now, let us prove that M is a good simulator for the view of B when interacting with prover A on input $(x, y) \in \mathcal{NQR}$. Actually, (A, B) is *perfect* zero-knowledge.

That is, the random variable output by M is the very same random variable seen by B (and thus the two random variables cannot be distinguished by any nonuniform algorithm, efficient or not). In fact, it can be easily seen that ρ is randomly distributed among the n^3-bit long strings. Moreover, if $\rho_i \in J_x^{+1}$, the corresponding s_i is a random (x,y)-root of ρ_i. Thus s_i has the same probability of belonging to M's output as it has of being sent from prover A to verifier B on inputs (x,y) and ρ. □

Note that the proof system (A,B) does not have *perfect* completeness; that is, there is a negligible probability that the prover, following the protocol, may not succeed in proving a true theorem. We can achieve perfect completeness and still retain perfect zero-knowledge at the expense of further complications which are not necessary in our context.

Robustness of the result. The above proof system is zero-knowledge if the reference string ρ is truly random. We may rightly ask what would happen if ρ is not truly randomly selected. Fortunately, we shall see that the poor randomness of ρ may perhaps weaken the zero-knowledgeness of our proof system, but not its completeness and soundness. In fact, all we require from ρ is that it contain a not too low percentage of quadratic residue and nonresidues modulo any integer in *Regular*(2) of a given length. The same remark applies to all proof systems of this paper. This robustness property is important, as we can never be sure of the quality of our natural sources of randomness.

5. A bounded noninteractive ZKPS for $3SAT$.

In this section we exhibit a bounded noninteractive ZKPS for $3SAT$. A boolean formula $\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ in conjunctive normal form over the variables $u_1, \cdots, u_k$, where each clause ϕ_i has three literals, is in the language $3SAT$ if it has a satisfying truth assignment $t: \{u_1, \cdots, u_k\} \to \{0,1\}$ (see [GaJo] for a more complete treatment). If $\Phi \in 3SAT$, we say that Φ is 3-satisfiable.

The following definition was informally introduced in [BlFeMi], but used in a quite different way.

DEFINITION 5.1. For any positive integer x, define the relation $\approx_x$ on $J_x^{+1} \times J_x^{+1} \times J_x^{+1}$ as follows:

$$(a_1,a_2,a_3) \approx_x (b_1,b_2,b_3) \iff a_i \sim_x b_i \quad \text{for } i=1,2,3.$$

Let $(a_1,a_2,a_3) \approx_x (b_1,b_2,b_3)$. An (a_1,a_2,a_3)-root modulo x (more simply, an (a_1,a_2,a_3)-root, when the modulus x is clear from the context) of (b_1,b_2,b_3) is a triplet (s_1,s_2,s_3) such that $(s_1^2 \bmod x,\ s_2^2 \bmod x,\ s_3^2 \bmod x) = (a_1b_1 \bmod x,\ a_2b_2 \bmod x,\ a_3b_3 \bmod x)$. If $\mathcal{Q}_x(b_1) = \mathcal{Q}_x(b_2) = \mathcal{Q}_x(b_3) = 0$, a square root modulo x (more simply a square root, when the modulus x is clear from the context) of (b_1,b_2,b_3) is a triplet (s_1,s_2,s_3) such that $(s_1^2 \bmod x, s_2^2 \bmod x, s_3^2 \bmod x) = (b_1,b_2,b_3)$.

From Fact 2.11, one can prove the following fact.

FACT 5.2. For each odd integer $x \in$ *Regular*(s), $\approx_x$ is an equivalence relation on $J_x^{+1} \times J_x^{+1} \times J_x^{+1}$ and there are $2^{3(s-1)}$ equally numerous $\approx_x$ equivalence classes.

We write $(a_1,a_2,a_3) \not\approx_x (b_1,b_2,b_3)$ when (a_1,a_2,a_3) is not $\approx_x$ equivalent to (b_1,b_2,b_3).

We now proceed as follows. In §5.1, we describe a sender–receiver pair (P,V). In §§5.2, 5.3, and 5.4 we will prove that (P,V) is a bounded noninteractive ZKPS for $3SAT$.

5.1. The sender–receiver pair (P,V).

Input to P and V:

- A random string $\rho \circ \tau$, where $|\rho| = 8n^3$ and $|\tau| = 2n^4$;
- $\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ a 3-satisfiable formula with n clauses over the variables $u_1, u_2, \cdots, u_k$, $k \leq 3n$.

Instructions for P.

P.1. Randomly select $x \in BL(n)$ and $y \in NQR_x$.

P.2. "Prove that $(x, y) \in \mathcal{NQR}(2n)$."
Send the *auxiliary pair* (x, y) and run algorithm A of §4 on inputs (x, y) and ρ. (Call $Proof_1$ the output.)

P.3. "Prove that $\Phi \in 3SAT$."
Let $t: \{u_1, \cdots, u_k\} \to \{0, 1\}$ be the lexicographically smallest satisfying assignment for Φ.
Execute procedure $\texttt{Prove}(\Phi, t, x, y, \tau)$ (see below). (Call $Proof_2$ the output.)

<u>**Procedure**</u> $\texttt{Prove}(\Phi, t, x, y, \tau)$

"$\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ is a 3-satisfiable formula with n clauses over the variables $u_1, u_2, \cdots, u_k$, $k \leq 3n$. $t: \{u_1, \cdots, u_k\} \to \{0, 1\}$ is a truth assignment satisfying Φ. $(x, y) \in \mathcal{NQR}(2n)$ and, moreover, $x \in BL(n)$. τ is a $2n^4$-bit long string."

begin{$\texttt{Prove}$}

1. "Break τ into members of J_x^{+1}."
Consider τ as the concatenation of n^3 $2n$-bit integers. If there are fewer than $33n^2$ integers in J_x^{+1} then stop. Else, let $\tau_1, \cdots, \tau_{33n^2}$ be the first $33n^2$ integers belonging to J_x^{+1}.
2. "Assign triplets of elements with Jacobi symbol $+1$ to clauses."
Group the τ_i's in $11n^2$ triplets $(\tau_1, \tau_2, \tau_3), (\tau_4, \tau_5, \tau_6), \cdots$. The first $11n$ triplets are *assigned* to ϕ_1, the second $11n$ triplets are *assigned* to ϕ_2, and so on.
3. "Label the formula Φ."
For each variable u_j, randomly select $r_j \in Z_x^*$ and compute the pairs (u_j, w_j) and $(\overline{u}_j, yw_j \bmod x)$, where
$$w_j = \begin{cases} r_j^2 \bmod x & \text{if } t(u_j) = 0, \text{ and} \\ yr_j^2 \bmod x & \text{if } t(u_j) = 1. \end{cases}$$
We refer to these pairs as the *labeling* of Φ and to w_j ($yw_j \bmod x$) as the *label* of the literal u_j ($\overline{u}_j$).
"Since y is a quadratic nonresidue, by Fact 2.1, yr_j^2 is a quadratic nonresidue. Therefore the label of a literal is a quadratic nonresidue if the literal is true under t."
Send the labeling of Φ.
4. "Prove that Φ is satisfiable."
For each clause ϕ of Φ do:
   - "Randomly select the verifying triplets."
   Let $(\alpha_1, \beta_1, \gamma_1)$ be the labels of the three literals of ϕ.
   Choose at random seven triplets $(\alpha_2, \beta_2, \gamma_2), \cdots, (\alpha_8, \beta_8, \gamma_8)$ in $J_x^{+1} \times J_x^{+1} \times J_x^{+1}$ such that
   (a) $(\alpha_i, \beta_i, \gamma_i) \not\approx_x (\alpha_j, \beta_j, \gamma_j)$ for $1 \leq i < j \leq 8$, and
   (b) $\mathcal{Q}_x(\alpha_2) = \mathcal{Q}_x(\beta_2) = \mathcal{Q}_x(\gamma_2) = 0$.
   Send $(\alpha_1, \beta_1, \gamma_1), \cdots, (\alpha_8, \beta_8, \gamma_8)$.

The triplets $(\alpha_1, \beta_1, \gamma_1), \cdots, (\alpha_8, \beta_8, \gamma_8)$ are the *verifying* triplets of ϕ. "We omit writing $(\alpha_1^\phi, \beta_1^\phi, \gamma_1^\phi), \cdots, (\alpha_8^\phi, \beta_8^\phi, \gamma_8^\phi)$ not to overburden our notation, hoping that clarity is maintained."

- "Prove that $(\alpha_2, \beta_2, \gamma_2)$ is made of quadratic residues."
  Randomly choose and send (s_1, s_2, s_3), a square root of $(\alpha_2, \beta_2, \gamma_2)$.
- For each of the assigned triplets (z_1, z_2, z_3) of ϕ, choose i, $1 \le i \le 8$, so that $(z_1, z_2, z_3) \approx_x (\alpha_i, \beta_i, \gamma_i)$. Randomly choose and send a $(\alpha_i, \beta_i, \gamma_i)$-root of (z_1, z_2, z_3).

end{`Prove`}

Instructions for V.

"V receives from P the auxiliary pair (x, y) and two strings $Proof_1$ and $Proof_2$."

V.0. Compute n from $\rho \circ \tau$ and verify that Φ has at most n clauses and each of them has three literals. If not, stop and REJECT. Else,

V.1. Run algorithm B of §4 on inputs ρ, (x, y), and $Proof_1$.
If B stops and rejects, stop and REJECT. Else,

V.2. If `Check_Prove`$(\Phi, x, y, \tau, Proof_2)$=ACCEPT then ACCEPT, else REJECT.

<u>**Procedure**</u> `Check_Prove`$(\Phi, x, y, \tau, Proof_2)$

"$\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ is a formula with n clauses over the variables $u_1, u_2, \cdots, u_k$. x, y are $2n$-bit integers. τ is a $2n^4$-bit long string. $Proof_2$ is a string."

begin{`Check_Prove`}

1. "Verify that the assigned triplets are proper."
   Consider τ as the concatenation of n^3 $2n$-bit integers. If there are fewer than $33n^2$ integers in J_x^{+1} stop and ACCEPT. Else, let $\tau_1, \cdots, \tau_{33n^2}$ be the first $33n^2$ integers belonging to J_x^{+1}.
   "This happens with very low probability."
   Group the τ_i's in $11n^2$ triplets $(\tau_1, \tau_2, \tau_3), (\tau_4, \tau_5, \tau_6), \cdots$. The first $11n$ triplets are *assigned* to ϕ_1, the second $11n$ triplets are *assigned* to ϕ_2, and so on. Verify that they have been properly computed by P.
2. "Verify that Φ has a proper labeling."
   For each variable u_j, verify that the label of the literal $\overline{u}_j$ is equal to the label of the literal u_j multiplied by y modulo x.
3. For each clause ϕ of Φ do:
   3.1. Let $(\alpha_i, \beta_i, \gamma_i)$, $i = 1, \cdots, 8$, be the verifying triplets of ϕ sent by P.
   3.2. Verify that $(\alpha_1, \beta_1, \gamma_1)$ is formed by the labels of the three literals of ϕ.
   3.3. Verify that (s_1, s_2, s_3) is a square root of $(\alpha_2, \beta_2, \gamma_2)$.
   3.4. Verify that for each assigned triplet (z_1, z_2, z_3) of ϕ, you received a $(\alpha_i, \beta_i, \gamma_i)$-root of (z_1, z_2, z_3), for some i, $1 \le i \le 8$.
4. If all the above verifications have been successfully made, return ACCEPT; otherwise return REJECT.

end{`Check_Prove`}

5.2. (P,V) is a bounded noninteractive proof system for $3SAT$. First, note that (P, V) is a sender–receiver pair. Further, all checks of V can be performed in polynomial time, since only simple algebraic computations modulo x and a scanning of the strings ρ and τ are needed.

Completeness. The same reasoning done in Theorem 4.3 shows that the probability that V does not REJECT at step V.1 is overwhelming. Let us now consider step V.2. The verification of the proper labeling of Φ is always passed. Since t is a satisfying truth assignment for Φ, each clause ϕ has at least one literal true under t. This implies that the label of ϕ contains at least one quadratic nonresidue. Because of this, and because there are eight $\approx_x$ equivalence classes, P can compute eight verifying triplets satisfying properties (a) and (b). Moreover, since each $\approx_x$ equivalent class contains a verifying triplet, each assigned triplet is $\approx_x$ equivalent to some $(\alpha_i, \beta_i, \gamma_i)$ and thus possesses an $(\alpha_i, \beta_i, \gamma_i)$-root. Therefore, if check V.1 is passed, so is check V.2.

Soundness. An honest prover chooses the pair (x, y) randomly. A cheating one, though, may choose this pair as function of the reference string. All arguments below thus have the following form. First, we compute the probability that the verifier can be mislead with a fixed pair, and show that this probability is suitably small. Then, we prove that, even summing up over all possible choices of pairs, we still obtain a small probability.

Assume that, in a computation with a cheating prover *Prover'*, V accepts a formula $\Phi \notin 3SAT$. Then, one of the following three events must happen: (a) the pair (x, y) chosen by *Prover'* is not in $\mathcal{NQR}(2n)$; (b) $(x, y) \in \mathcal{NQR}(2n)$, but *Prover'* stops at step P.1 in `Prove`; and (c) $(x, y) \in \mathcal{NQR}(2n)$, *Prover'* does not stop at step P.1 in `Prove`, but Φ is not 3-satisfiable. We shall prove that each of these events is very improbable. The probability that (a) occurs has already been computed in Theorem 4.3 and shown to be exponentially vanishing in n. Now, consider event (b). For each fixed $\overline{x} \in \mathit{Regular}(2), \overline{x} \leq n$, since each τ_i has probability greater than or equal to $\frac{1}{8}$ of being in $J_{\overline{x}}^{+1}$, we expect $n^3/8$ such elements in $J_{\overline{x}}^{+1}$. By the Chernoff bound (see [AnVa], [ErSp]), the probability that no more than $33n^2$ belong to $J_{\overline{x}}^{+1}$ is, for large n, at most e^{-n^2}. Thus, the probability that there is an integer x such that case (b) occurs is, for large n, at most $2^{2n}e^{-n^2}$. Let us now consider event (c). If (c) occurs, then the following event (d) must also occur: at least $11n$ consecutive assigned triplets $(\tau_i, \tau_{i+1}, \tau_{i+2})$ must belong to the union of seven $\approx_x$ equivalence classes. In fact, if Φ is not satisfiable, for every labeling of Φ, one of its clauses is labeled with a triplet of quadratic residues (else, all clauses would be satisfiable). Let ϕ be such a clause. Since verification step 3.3 must be passed, *Prover'* must exhibit a square root of $(\alpha_2, \beta_2, \gamma_2)$, and thus this triplet is $\approx_x$ equivalent to ϕ's label, $(\alpha_1, \beta_1, \gamma_1)$. Thus, all verifying triplets of ϕ are contained in the union of at most seven $\approx_x$ equivalence classes. Since each $(\tau_i, \tau_{i+1}, \tau_{i+2})$ is proved in step 3.4 to be $\approx_x$ equivalent to one verifying triplet, then event (d) must be true. The probability of event (d) is at most $7n(0.93)^n$. Indeed, for each fixed $\overline{x}$ the probability that at least $11n$ assigned triplets belong to the union of 7 $\approx_{\overline{x}}$ equivalence classes is less than $7n(\frac{7}{8})^{11n}$; this can be explained as follows: $\frac{7}{8}$ is the probability that each triplet belongs to the union of seven fixed equivalence classes, there are $11n$ triplets, there are at most $\binom{8}{7} = 8$ ways to choose seven classes out of eight, and there are n clauses altogether. Therefore, the probability that there exists an integer x such that case (d) occurs is at most $2^{2n}7n(\frac{7}{8})^{11n} < 7n(0.93)^n$. This concludes the proof of soundness.

Remark. (P, V) can be modified in the same way as (A, B) was to achieve perfect completeness. This is the reason why the verifier in step 1 of `Check_Prove` accepts if there are fewer than $33n^2$ integers in J_x^{+1}. Note also that the prover need not have infinite computing power. In fact, an efficient algorithm can perform all required computations provided that it has as an additional input the satisfying assignment for Φ.

We show now that the proof system (P, V) is also zero-knowledge over $3SAT$.

We first exhibit a simulator for V's view and then prove that it works.

5.3. The simulator. The following algorithm S, on input a formula $\Phi \in 3SAT$ (but not a satisfying assignment for Φ) generates a family of random variables that, under the QRA, no efficient nonuniform algorithm can distinguish from the view of V. Note that the view of V consists of a quadruple $(\rho \circ \tau, (x, y), Proof_1, Proof_2)$; thus, the task of the simulator is to produce a quadruple that cannot be distinguished, under the QRA, from a correct quadruple. Looking ahead, the two crucial points in the strategy of the simulator are:

1. To choose the auxiliary pair (x, y) so that $x \in BL(n)$ but y is a quadratic residue modulo x.
2. To choose a portion of the reference string not at random. Rather, select it from among the strings that do not contain any quadratic nonresidue modulo x in J_x^{+1}.

This strategy is viable because the simulator can choose the reference string (which is instead fixed for the prover) and because it is hard to distinguish between random members of J_x^{+1} and random quadratic residues modulo x.

For a clearer presentation, S's program has been broken down into procedures. To give informal help in reading these procedures, we write z' for a value computed by the simulator, when we want to emphasize that this value is "fundamentally different" from the "corresponding" value z computed by the prover P, though an exponentially long computation may be required to determine this fact.

S's program

Input: a 3-satisfiable formula $\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ over the variables $u_1, u_2, \cdots, u_k$, $k \leq 3n$.

1. Randomly select two n-bit primes $p, q \equiv 3 \bmod 4$ and set $x = pq$. Randomly select $r \in Z_x^*$ and set $y' = r^2 \bmod x$. "Call (x, y') the *auxiliary pair*."
2. Execute procedure Gen_ρ_and_Proof1(x, y') obtaining the strings ρ' and $Proof_1$.
3. Generate a random $2n^4$-bit string τ.
4. Execute procedure Gen_Proof2$(\Phi, x, y', p, q, \tau)$ obtaining the string $Proof_2$.

Output: $(\rho' \circ \tau, (x, y'), Proof_1, Proof_2)$

__Procedure__ Gen_ρ_and_Proof1(x, y)

"This procedure is used both by the simulator S and, later on, by some probabilistic algorithm. In any call, $x \in BL(n)$ and $y \in J_x^{+1}$. When the procedure is called by the simulator S, y is a quadratic residue modulo x."

begin{Gen_ρ_and_Proof1}

1. Set $Proof_1$ = empty string.
2. For $i = 1$ to $4n^2$
   Randomly select a $2n$-bit integer s_i, with possible leading 0's.
   If $s_i \notin J_x^{+1}$ then set $\rho_i = s_i$.
   else
   Toss a fair coin.
   If HEAD then set $\rho_i = s_i^2 \bmod x$ and append s_i to $Proof_1$.
   If TAIL then set $\rho_i = y^{-1} s_i^2 \bmod x$ and append $s_i \bmod x$ to $Proof_1$.
3. Set $\rho = \rho_1 \cdots \rho_{4n^2}$.
4. Return$(\rho, Proof_1)$

end{Gen_ρ_and_Proof1}

Let us now see that sometimes Gen_ρ_and_Proof1 "generates what the legitimate prover would generate."

LEMMA 5.3. *Define* Space1(x, y) *as the probability space generated by the output of* Gen_ρ_and_Proof1 *on input* x, y. *Then, for all* $x \in BL(n)$ *and* $y \in NQR_x$

$$\texttt{Space1}(x,y) = \left\{\rho \overset{R}{\leftarrow} \{0,1\}^{8n^3}; Proof_1 \overset{R}{\leftarrow} P\_Proof1(x,y,\rho) : (\rho, Proof_1)\right\},$$

where P_Proof1 *is* P*'s procedure to compute* $Proof_1$ *(i.e., step P.2).*

Proof. Fix $x \in BL(n)$ and $y \in NQR_x$. It can easily be seen that the first component of Gen_ρ_and_Proof1's output is randomly distributed among the $8n^3$-bit long strings. Moreover, if $\rho_i \in J_x^{+1}$, the corresponding s_i is a random (x,y)-root of ρ_i. Thus s_i has the same probability of belonging to Gen_ρ_and_Proof1's output as it has of being sent, at step P.2, from prover P to verifier V on inputs (x, y) and ρ. □

Procedure Gen_Proof2$(\Phi, x, y', p, q, \tau)$

"This procedure is used both by the simulator S and, later on, by some probabilistic algorithm. In any call, $x \in BL(n)$ and $y' \in QR_x$. It returns a string $Proof_2$ that 'proves' that the formula $\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ is 3-satisfiable using the string τ and the pair (x, y') even without knowing any satisfying assignment for Φ."

begin{Gen_Proof2}

0. Set $Proof_2$ = empty string.
1. Consider τ as the concatenation of n^3 $2n$-bit integers. If there are fewer than $33n^2$ integers in J_x^{+1}, stop. Else, let $\tau_1, \cdots, \tau_{33n^2}$ be the first $33n^2$ integers belonging to J_x^{+1}.
   Group the τ_i's in $11n^2$ triplets $(\tau_1, \tau_2, \tau_3), (\tau_4, \tau_5, \tau_6), \cdots$. The first $11n$ triplets are *assigned* to ϕ_1, the second $11n$ triplets are *assigned* to ϕ_2, and so on.
2. For each variable u_j, randomly select $w_j \in NQR_x$ and label the literal u_j with w_j and the literal $\overline{u}_j$ with $y'w_j \bmod x$.
   "Since y' is a quadratic residue, all labels are quadratic nonresidues."
   Append the labeling of Φ to $Proof_2$.
3. For each clause ϕ of Φ do:
   - Let α_1, β_1, and γ_1 be the labels of the three literals of ϕ. Thus, $\alpha_1, \beta_1, \gamma_1 \in NQR_x$.
     Choose at random seven triplets $(\alpha_2, \beta_2, \gamma_2), \cdots, (\alpha_8, \beta_8, \gamma_8)$ in $J_x^{+1} \times J_x^{+1} \times J_x^{+1}$ such that $(\alpha_i, \beta_i, \gamma_i) \not\approx_x (\alpha_j, \beta_j, \gamma_j)$, for $1 \le i < j \le 8$ and $\mathcal{Q}_x(\alpha_2) = \mathcal{Q}_x(\beta_2) = \mathcal{Q}_x(\gamma_2) = 0$.
     Append the triplets $(\alpha_1, \beta_1, \gamma_1), \cdots, (\alpha_8, \beta_8, \gamma_8)$ as the *verifying* triplets of ϕ to $Proof_2$.
   - Randomly choose and append a square root of $(\alpha_2, \beta_2, \gamma_2)$ to $Proof_2$.
   - For each of the assigned triplets (z_1, z_2, z_3) of ϕ, choose i, $1 \le i \le 8$, so that $(z_1, z_2, z_3) \approx_x (\alpha_i, \beta_i, \gamma_i)$. Randomly choose and append an $(\alpha_i, \beta_i, \gamma_i)$-root of (z_1, z_2, z_3) to $Proof_2$.
4. Return($Proof_2$)

end{Gen_Proof2}

LEMMA 5.4. *Algorithm* S *is efficient.*

Proof. The main body and procedure Gen_ρ_and_Proof1 are computationally trivial. The first two steps of procedure Gen_Proof2 are also quite easy as, due to

Fact 2.9, generating a random quadratic nonresidue in J_x^{+1} is easy when $x \in BL$. Let us now see also that step 3 can always be completed, and efficiently as well. Given that the first verifying triplet has been chosen to be composed by quadratic nonresidues in J_x^{+1} and the second by quadratic residues, it is certainly possible to choose the other six verifying triplets so that all of them belong to eight distinct $\approx_x$ equivalence classes. Moreover, given that the factorization of x is an available input, the remaining part of step 3 can be efficiently executed. □

5.4. (P,V) is zero-knowledge.

THEOREM 5.5. *Under the* QRA, (P, V) *is a bounded noninteractive* ZKPS *for* $3SAT$.

Proof. All that is left to prove is that (P, V) satisfies the zero-knowledge condition. We do this by showing that algorithm S of the previous section simulates the view of the verifier V.

We proceed by contradiction. Assume that there exists a positive constant d, an infinite subset $\mathcal{I} \subseteq \mathcal{N}$, a set $\{\Phi_n\}_{n\in\mathcal{I}}$ such that each Φ_n is a 3-satisfiable formula with n clauses, and an efficient nonuniform "distinguishing" algorithm $\{D_n\}_{n\in\mathcal{I}}$ such that for all $n \in \mathcal{I}$

$$\left|P_S(n) - P_V(n)\right| \geq n^{-d},$$

where

$$P_S(n) = Pr(s \stackrel{R}{\leftarrow} S(1^n, \Phi_n) : D_n(s) = 1)$$

and

$$P_V(n) = Pr(s \stackrel{R}{\leftarrow} View(\Phi_n) : D_n(s) = 1).$$

We derive a contradiction by showing an efficient nonuniform algorithm $\{C_n\}_{n\in\mathcal{I}}$ violating the QRA. On input randomly chosen $x \in BL(n)$ and $y \in J_x^{+1}$, C_n constructs a string $SAMPLE$ which is distributed according to $S(1^n, \Phi_n)$ if $y \in QR_x$, and according to $View(\Phi_n)$ if $y \in NQR_x$. Thus, as the nonuniform algorithm $\{D_n\}_{n\in\mathcal{I}}$ is assumed to distinguish the two probability spaces, this is a violation of QRA.

The Algorithm C_n

"C_n has "wired-in" a formula Φ_n along with t, the lexicographically smaller satisfying truth assignment for Φ_n, a description of D_n, and the probabilities $P_S(n)$ and $P_V(n)$."

Input: (x, y) such that $x \in BL(n)$ and $y \in J_x^{+1}$.

1. Execute procedure $\texttt{Gen\_}\rho\texttt{\_and\_Proof1}(x, y)$, thus obtaining ρ and $Proof_1$.
2. Execute procedure $\texttt{Sample\_}\tau\texttt{\_and\_Proof2}(\Phi_n, t, x, y)$, thus obtaining τ and $Proof_2$.
3. Set $SAMPLE = \left(\rho \circ \tau, (x, y), Proof_1, Proof_2\right)$.
4. If $D_n(SAMPLE) = 1$ then set $b = 1$ else $b = 0$.
5. If $P_S(n) > P_V(n)$ then **Output**(b) else **Output**$(1 - b)$.

<u>**Procedure**</u> $\texttt{Sample\_}\tau\texttt{\_and\_Proof2}(\Phi, t, x, y)$

"$\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ is a 3-satisfiable formula with n clauses over the variables $u_1, u_2, \cdots, u_k$, $k \leq 3n$. $t : \{u_1, u_2, \cdots, u_k\} \to \{0, 1\}$ is a satisfying truth assignment for Φ. $x \in BL(n)$ and $y \in J_x^{+1}$."

begin{Sample_τ_and_Proof2}

1. For $i = 1$ to n^3 do:
   randomly select a $2n$-bit integer r_i (with possible leading 0's)
   if $r_i \notin J_x^{+1}$ then set $s_i = r_i$
   else toss a fair coin: if HEAD then set $s_i = r_i^2 \bmod x$; if TAIL then set $s_i = -r_i^2 \bmod x$.
2. Set $Proof_2$= empty string.
3. Let $j_1, \cdots, j_{33n^2}$ be the indices of the first $33n^2$ s_i's belonging to J_x^{+1}.
   If there are fewer than $33n^2$ such integers set $\tau = s_1 \cdots s_{n^3}$ and stop.
   Else, set $\tau_i = s_i$ for all indices i not in $\{j_1, \cdots, j_{33n^2}\}$.
4. Group the j_i's in $11n^2$ triplets $(j_1, j_2, j_3), (j_4, j_5, j_6), \cdots$. Assign the $11n^2$ triplets to the clauses in the following way: the first $11n$ triplets are *assigned* to the first clause, ϕ_1, the second $11n$ triplets are *assigned* to the second clause, ϕ_2, and so on.
5. For each variable u_j, randomly select $v_j \in Z_x^*$ and assign the *label* w_j to the literal u_j and the label $yw_j \bmod x$ to the literal $\overline{u}_j$, where

$$w_j = \begin{cases} -v_j^2 \bmod x & \text{if } t(u_j) = 1, \text{ and} \\ -yv_j^2 \bmod x & \text{if } t(u_j) = 0. \end{cases}$$

   Call Φ' the labeling of Φ. Append Φ' to $Proof_2$.
6. For each clause ϕ of Φ do:
   - Let $-ya^2 \bmod x, -yb^2 \bmod x, -c^2 \bmod x$ be the label of the three literals of ϕ, and a, b, c previously computed values in Z_x^*.
     "We consider only one case, not to overburden our notation. The other cases are treated similarly."
   - Randomly choose 21 elements $a_1, b_1, c_1, \cdots, a_7, b_7, c_7 \in Z_x^*$, and construct the following eight triplets

$$\begin{gathered}(-ya^2 \bmod x, -yb^2 \bmod x, -c^2 \bmod x) \\ (a_1^2 \bmod x, b_1^2 \bmod x, c_1^2 \bmod x) \\ (a_2^2 \bmod x, -b_2^2 \bmod x, c_2^2 \bmod x) \\ (a_3^2 \bmod x, -b_3^2 \bmod x, -c_3^2 \bmod x) \\ (-a_4^2 \bmod x, b_4^2 \bmod x, c_4^2 \bmod x) \\ (-a_5^2 \bmod x, b_5^2 \bmod x, -c_5^2 \bmod x) \\ (-a_6^2 \bmod x, -b_6^2 \bmod x, c_6^2 \bmod x) \\ (ya_7^2 \bmod x, yb_7^2 \bmod x, -c_7^2 \bmod x).\end{gathered}$$

   - Construct the eight *verifying* triplets of ϕ as follows. Set

$$\begin{aligned}(\alpha_1, \beta_1, \gamma_1) &= (-ya^2 \bmod x, -yb^2 \bmod x, -c^2 \bmod x), \\ (\alpha_2, \beta_2, \gamma_2) &= (a_1^2 \bmod x, b_1^2 \bmod x, c_1^2 \bmod x).\end{aligned}$$

     Randomly permute the remaining six triplets and assign them to

$$(\alpha_3, \beta_3, \gamma_3), \cdots, (\alpha_8, \beta_8, \gamma_8).$$

     Append $(\alpha_1, \beta_1, \gamma_1), \cdots, (\alpha_8, \beta_8, \gamma_8)$ to $Proof_2$.
   - Append the triplet (a_1, b_1, c_1) to $Proof_2$ as a square root of $(\alpha_2, \beta_2, \gamma_2)$.
   - For each of the assigned indices (l_1, l_2, l_3) of ϕ,
     Randomly choose one of the eight verifying triplets, say, $(\alpha_k, \beta_k, \gamma_k)$.
     Randomly choose $v_1, v_2, v_3 \in Z_x^*$ and set $\tau_{l_1} = v_1^2 \alpha_k \bmod x$, $\tau_{l_2} = v_2^2 \beta_k \bmod x$, and $\tau_{l_3} = v_3^2 \gamma_k \bmod x$.

Compute and append to $Proof_2$ $(v_1\alpha_k \bmod x, v_2\beta_k \bmod x, v_3\gamma_k \bmod x)$ as an $(\alpha_k, \beta_k, \gamma_k)$-root of $(\tau_{l_1}, \tau_{l_2}, \tau_{l_3})$.

- Set $\tau = \tau_1 \cdots \tau_{n^3}$.

7. Return(τ, $Proof_2$).

end{`Sample_τ_and_Proof2`}

There is no question that $\{C_n\}_{n\in\mathcal{I}}$ is an efficient nonuniform algorithm. Now let $\mathtt{Space2}(\Phi_n, t, x, y)$ be the probability space generated by the output of `Sample_τ_and_Proof2` on input Φ_n, t, x, y. Then, for all $n \in \mathcal{I}$ and for all $x \in BL(n)$, $\mathtt{Space2}(\Phi_n, t, x, y)$ is equal to

$$(*)\begin{cases} \{\tau \xleftarrow{R} \{0,1\}^{2n^4}; Proof_2 \xleftarrow{R} \mathtt{Prove}(\Phi_n, t, x, y, \tau) : (\tau, Proof_2)\} \text{ if } y \in NQR_x, \\ \{\tau \xleftarrow{R} \{0,1\}^{2n^4}; Proof_2 \xleftarrow{R} \mathtt{Gen\_Proof2}(\Phi_n, x, y, p, q, \tau) : (\tau, Proof_2)\} \text{if } y \in QR_x, \end{cases}$$

where p, q are the prime factors of x.

To see $(*)$, note that if $y \in NQR_x$, then the label w_j assigned to each literal u_j by C_n is a random element selected from either NQR_x or QR_x depending on whether $t(u_j)$ is true or false, respectively (this is the same computation performed by `Prove`). If $y \in QR_x$, then the label w_j of literal u_j is always a random element selected from NQR_x (in the same way as `Gen_Proof2` computes it). In both cases the label of $\overline{u}_j$ is $yw_j \bmod x$.

Regardless of the quadratic residuosity of y modulo x, for each clause ϕ of Φ, the eight verifying triplets of ϕ computed by C_n are always selected at random among the triplets of elements in J_x^{+1} that are pairwise not $\approx_x$ equivalent. The first triplet consists of the labels of the three literals of ϕ, and the second triplet is made of three quadratic residues.

The string τ output by C_n is truly random (regardless of the quadratic residuosity of y modulo x). Indeed, each τ_i is randomly selected from the $2n$-bit long strings, and independently of the remaining τ_j's.

Finally, for each clause and each of its assigned triplets $(\tau_{l_1}, \tau_{l_2}, \tau_{l_3})$ the corresponding $(v_1\alpha_k \bmod x,\ v_2\beta_k \bmod x,\ v_3\gamma_k \bmod x)$ is a random $(\alpha_k, \beta_k, \gamma_k)$-root of $(\tau_{l_1}, \tau_{l_2}, \tau_{l_3})$. This completes the proof of $(*)$.

Since $SAMPLE = (\rho \circ \tau, (x, y), Proof_1, Proof_2)$, because of $(*)$ and because of Lemma 5.3, for randomly selected $x \in BL(n)$ and $y \in J_x^{+1}$, $SAMPLE$ is distributed as $View(\Phi_n)$ if $y \in NQR_x$ and as $S(1^n, \Phi_n)$ if $y \in QR_x$. Given our assumption about the efficient nonuniform algorithm $\{D_n\}_{n\in\mathcal{I}}$, it is immediately seen that, for all $n \in \mathcal{I}$, $Pr\big(x \xleftarrow{R} BL(n);\ y \xleftarrow{R} J_x^{+1} : C_n(x, y) = \mathcal{Q}_x(y)\big) \geq 1/2 + 1/(2n^d)$, which contradicts the QRA. □

Remark. The reader is encouraged to verify that if the same reference string σ and the same (x, y) are used by the prover to prove that two formulae Φ and $\hat{\Phi}$ are 3-satisfiable, then "extra knowledge may leak," for instance, that there exist a satisfying assignment for Φ and a satisfying assignment for $\hat{\Phi}$ for which the literal u_1 in Φ and the literal $\hat{u}_2$ in $\hat{\Phi}$ have the same truth value.

The moral is that one must be careful when using the same set-up, i.e., common reference string, and the same pair (x, y), to prove an "unlimited" number of formulae to be satisfiable. This is indeed the goal of §6.

5.5. Arthur–Merlin games and bounded noninteractive zero knowledge.

THEOREM 5.6. *If $3SAT \in$ Bounded-NIZK, then Bounded-NIZK $= AM_2$.*

Proof. Since *Bounded*-NIZK $\subseteq$ *Bounded noninteractive* $P = AM_2$, it only remains to show that $AM_2 \subseteq$ *Bounded*-NIZK. Let $L \in AM_2$. Then, there exist a positive constant c and a sender–receiver pair (*Prover, Verifier*) such that

1. for all $x \in L_n$,

$$Pr(\sigma \stackrel{R}{\leftarrow} \{0,1\}^{n^c}; Proof \stackrel{R}{\leftarrow} Prover(\sigma, x) : \ Verifier(\sigma, x, Proof) = 1) > \frac{2}{3}$$

and

2. for all $x \notin L_n$, for all Turing machines $Prover'$, and for all sufficiently large n,

$$Pr(\sigma \stackrel{R}{\leftarrow} \{0,1\}^{n^c}; Proof \stackrel{R}{\leftarrow} Prover'(\sigma, x) : \ Verifier(\sigma, x, Proof) = 1) < \frac{1}{3}.$$

Moreover, by the result of [FuGoMaSiZa], the proof system (*Prover, Verifier*) enjoys perfect completeness. Define now the language $L' = \cup_n L'(n)$, where

$$L'(n) = \{(r, x) : |r| = n^c, \ x \in L_n, \text{ and } \exists w, \ |w| \leq n^c \text{ such that } Verifier(r, x, w) = 1\}$$

and L and c are as above. Then $x \in L_n$ if and only if $(r, x) \in L'(n)$ for most n^c-bit strings r.[7] Moreover, $L' \in NP$, thus there is a fixed polynomial-time computable reduction R such that

$$(r, x) \in L'(n) \iff \Psi = R(r, x) \in 3SAT_{n^b},$$

where $b > 0$ is a fixed constant depending only on the reduction R.

We now describe a bounded noninteractive ZKPS (P, V) for L. On input $x \in L_n$ and the reference string $\tau = r \circ \sigma$, where $|r| = n^c$ and σ has the proper length, P constructs the formula $\Psi = R(r, x)$ and, if it is 3-satisfiable, then runs the algorithm for the prover P of §5.1 with input Ψ and σ, to prove that, indeed, $\Psi \in 3SAT_{n^b}$. □

THEOREM 5.7. *Under the* QRA, *Bounded*-NIZK $= AM_2$.

6. Noninteractive zero-knowledge. We now want to capture the ability of giving noninteractive and zero-knowledge proofs of "many" theorems, using the same common reference string, in an "on-line manner." That is, each theorem can be proven independently of all previous and future theorems.

We will present our formal definition when the theorems to be proven are statements about 3-satisfiability.

DEFINITION 6.1. Let (*Prover, Verifier*) be a sender–receiver pair, where $Prover(\cdot,\cdot)$ is random selecting and $Verifier(\cdot,\cdot,\cdot)$ is polynomial time. We say that (*Prover, Verifier*) is a noninteractive zero-knowledge proof system (noninteractive ZKPS) if the following three conditions hold.

1. *Completeness.* For all $\Phi \in 3SAT$ and all n,

$$Pr\Big(\sigma \stackrel{R}{\leftarrow} \{0,1\}^n; Proof \stackrel{R}{\leftarrow} Prover(\sigma, \Phi) : \ Verifier(\sigma, \Phi, Proof) = 1\Big) = 1.$$

[7] Thus an alternative way of proving that $x \in L_n$ consists of showing that, for a random string r of the proper length, $(r, x) \in L'(n)$. Note, though, that there may be two different strings x and y in L_n such that $(r, x) \in L'(n)$ for all r, but $(r, y) \notin L'(n)$ for some r's. Thus the fact that for a given string r, $(r, x) \in L'(n)$ constitutes additional information about x than just membership in L_n, and this additional information cannot be hidden by a zero-knowledge proof that $(r, x) \in L'(n)$! This is why we impose the conditions that (*Prover, Verifier*) possess perfect completeness.

2. *Soundness.* There exists a constant $c_1 > 0$ such that, for all probabilistic algorithms *Adversary* outputting pairs $(\Phi', Proof')$, where $\Phi' \notin 3SAT$, for all $d > 0$, and for all $n > c_1$,

$$Pr\Big(\sigma \overset{R}{\leftarrow} \{0,1\}^n; (\Phi', Proof') \overset{R}{\leftarrow} Adversary(\sigma): \ Verifier(\sigma, \Phi', Proof') = 1\Big) < n^{-d}.$$

3. *Zero-knowledge.* There exist constant $c_2 > 0$, an efficient algorithm S such that for all $\Phi_1, \Phi_2, \cdots \in 3SAT$, for all efficient nonuniform algorithms D, for all $d > 0$, and all $n > c_2$,

$$\big|Pr(s \overset{R}{\leftarrow} View(n, \Phi_1, \Phi_2, \cdots) : D_n(s) = 1) - Pr(s \overset{R}{\leftarrow} S(1^n, \Phi_1, \Phi_2, \cdots) : D_n(s) = 1)\big| < n^{-d}$$

where

$$View(n, \Phi_1, \Phi_2, \cdots) = \Big\{\sigma \overset{R}{\leftarrow} \{0,1\}^n; \begin{array}{rl} Proof_1 & \overset{R}{\leftarrow} Prover(\sigma, \Phi_1); \\ Proof_2 & \overset{R}{\leftarrow} Prover(\sigma, \Phi_2); \\ & \vdots \\ & : (\sigma, Proof_1, Proof_2, \cdots)\Big\}. \end{array}$$

A sender–receiver pair (*Prover, Verifier*) is a noninteractive proof system for $3SAT$ if completeness and soundness hold.

Discussion. First, note that we have set the probability of acceptance of true theorems to be 1, since $3SAT \in NP$. Note also the generality of our definition as it handles any number of formulae of arbitrary size in completeness, soundness, and zero-knowledge. That is, every true theorem can be proven, no matter how long. Of course, longer theorems will have longer proofs. Since the verifier is polynomial-time in the length of the common input, it will have more time to verify that a longer formula is 3-satisfiable. Every false theorem, no matter how long, has negligible probability of being "successfully proved"; however, though the length of the proof grows with the length of the theorem, "negligible" is defined only as a function of the length of the reference string.[8] Finally, every theorem, no matter how long, possesses a zero-knowledge proof. Of course, a longer theorem will have a longer proof and thus the polynomial-time simulator will have more time to simulate the proofs. The zero-knowledgeness of the simulator's proofs holds only for a nonuniform "observer" bounded by the length of the reference string.[9]

The definition of noninteractive ZKPS might be more general if perfect completeness is relaxed to completeness as in §3. In this case the adversary choosing algorithm *Choose-in-L* should be given σ and access to *Prover*'s random selector.

6.1. The sender–receiver pair (P,V). In this subsection we describe a sender-receiver pair (P, V). P can prove in zero-knowledge the 3-satisfiability of any number of 3-satisfiable formulae with n clauses each. Later, we shall show how to use the same protocol to prove any number of formulae, each of arbitrary size.

Before going into a formal description of the proof system, we give an informal view of the protocol.

[8] Which, de facto, is a security parameter.

[9] In particular, if a theorem and its proof are exponentially long (with respect to the reference string), the distinguishing algorithm can compare the actual "view" and the output of the simulator only for a polynomially long prefix.

An informal look at (P,V).

Observation. A crucial observation that will be (implicitly) proved in this section is the following. If many certified auxiliary pairs (x, y) ($x \in BL$ and $y \in NQR_x$) are available, one can use each (x, y) to prove in zero-knowledge that any *single* formula $\Phi_{(x,y)}$ with n clauses is 3-satisfiable using the *same* random string τ. For what we remarked in §5, the same τ and the same auxiliary pair should not be used to prove the 3-satisfiability of two different formulae.

In the light of the above observation, we want to construct a mechanism to achieve the following two goals:

(1) Associating to each formula Φ an auxiliary pair (x^Φ, y^Φ), of "bounded" size, so that, with overwhelming probability, different formulae are associated to different pairs.

(2) Certifying (x^Φ, y^Φ), i.e., proving that $x^\Phi \in BL$ and $y^\Phi \in NQR_{x^\Phi}$.

The first goal could be achieved by using the random selector, but the problem of the certification remains. The current mechanism for certifying in zero-knowledge a single auxiliary pair (x, y) using ρ can be extended to handle "a few" more pairs, but not arbitrarily many.[10] Instead, we use a mechanism of recursive nature to simultaneously achieve (1) and (2).

Let us first describe this recursive mechanism for a prover "with memory." Such a prover can construct and store a binary tree of depth n. The left child of each node will also be denoted as the 0-child, and the right one as the 1-child. Thus each node in the tree is labeled with a binary string of length at most $n+1$. The root is labeled 0, and each other node is labeled with string describing the unique path from the root to it. Thus, for instance, the left child of the root has label 00 and rightmost leaf of the tree has label 01^n. With each node (labeled) i, the prover stores a randomly selected auxiliary pair (x_i, y_i). The prover uses (x_i, y_i) for certifying auxiliary pairs of the children of node i, that is, (x_{i0}, y_{i0}) and (x_{i1}, y_{i1}). The first auxiliary pair (x_0, y_0) is certified using string ρ as in §4. For each i, the two pairs $(x_{0b_1\cdots b_i 0}, y_{0b_1\cdots b_i 0})$, $(x_{0b_1\cdots b_i 1}, y_{0b_1\cdots b_i 1})$, are certified together as in §5, using the same string τ_1. That is, consider the language $L = \cup_n L(n)$, where

$$L(n) = \{((u_0, v_0), (u_1, v_1)) : u_0, u_1 \in BL(n),\ v_0 \in NQR_{u_0},\ v_1 \in NQR_{v_1}\}.$$

Then $L \in NP$. Thus, there exists a fixed polynomial-time computable function CR such that

$$((u_0, v_0), (u_1, v_1)) \in L(n) \iff \Psi = CR(u_0, v_0, u_1, v_1) \in 3SAT_{n^e},$$

where e is a fixed constant depending only on the reduction CR. More precisely, let T be a polynomial-time Turing machine such that $x \in L$ if and only if there is a "witness" (string) w such that $|w| \le |x|^e$ and $T(x, w) = 1$. Then, the formula Ψ is obtained by encoding the computation of T as in Cook's theorem, and then reducing it to a 3-satisfiable formula, as Cook suggested [Co]. A well-known property of this reduction is that to each "witness" w one can associate in polynomial time a satisfying assignment for Ψ. In our case the witness consists of the primes in the factorizations of u_0 and u_1 and their proof of primality. The proof (witness) of the primality of

[10] Recall the way ρ is used. If $\rho_i \in QR_x$, a square root of ρ_i mod x is given; if $\rho_i \in NQR_x$ a square root of $y\rho_i$ mod x is given. In our simulation, however, all ρ_i will be chosen in QR_x. Thus, if we want to carry on the simulation for many pairs (x_i, y_i) we need to construct a ρ solely consisting of quadratic residues modulo $x_1, x_2, \cdots$, which appears very hard to do when the number of x_i's grows large.

a prime p is probabilistically constructed in a standard way: by running algorithm [AdHu] on input p, flipping coins as needed.

We will thus certify $(x_{0b_1\cdots b_i0}, y_{0b_1\cdots b_i0})$, $(x_{0b_1\cdots b_i1}, y_{0b_1\cdots b_i1})$ by showing that the so constructed

$$\Psi_{0b_1\cdots b_i} = CR((x_{0b_1\cdots b_i0}, y_{0b_1\cdots b_i0}), (x_{0b_1\cdots b_i1}, y_{0b_1\cdots b_i1})) \in 3SAT_{n^e}.$$

For each $\Psi_{0b_1\cdots b_i}$, this is done using the proof system of §5, and the *same* string τ_1, which in fact has length $2n^a$, with $a = 4e$.

What have we gained by this? Essentially, we have transformed the problem of *certifying* $(x_{0b_1\cdots b_i0}, y_{0b_1\cdots b_i0})$, $(x_{0b_1\cdots b_i1}, y_{0b_1\cdots b_i1})$ into the problem of *proving* that $\Psi_{0b_1\cdots b_i} \in 3SAT_{n^e}$, and we have observed (but not yet proved) that one can prove in zero-knowledge arbitrarily many theorems of size n given arbitrarily many independent certified pairs (x, y)'s. Since these pairs are randomly and independently selected, with overwhelming probability, each pair $(x_{0b_1\cdots b_i}, y_{0b_1\cdots b_i})$ is used only once with τ_1 to prove $\Psi_{0b_1\cdots b_i} \in 3SAT_{n^e}$.

In sum, this mechanism provides each formula Φ with a certified auxiliary pair (x^Φ, y^Φ) that is uniquely determined from Φ and the reference string, though still random.

The prover we just described need not remember the labeled full binary tree; it can, in fact, (re)grow its branches as needed. It must, though, remember which auxiliary pairs it had associated with the nodes of the tree. In fact, if it does not keep track of these pairs, it may use the same auxiliary pair and the same reference string to prove different theorems, which may not be zero-knowledge. To avoid this, and to avoid "memory," the prover uses the random selector to associate a random pair with the node of the tree. Namely, on input a formula Ψ the prover chooses n bits $b_1b_2\cdots b_n$ by querying the random selector with a pair whose first entry is Ψ and the reference string $\sigma = \rho\circ\tau_1\circ\tau_2$, and whose second entry is (a description of) the set $\{0,1\}^n$. This way, if the same formula is considered twice, the same random n-bit string would be selected. Then the prover computes a random, first auxiliary pair (x_0, y_0) (again using the random selector so that it could recompute the same pair whenever it wanted to). Then, for $i = 0, \cdots, n$, the auxiliary pairs $(x_{0b_1\cdots b_i0}, y_{0b_1\cdots b_i0})$, $(x_{0b_1\cdots b_i1}, y_{0b_1\cdots b_i1})$, are chosen by the random selector on input $0b_1\cdots b_i0$ and $0b_1\cdots b_i1$, respectively. The pair associated with Φ is $(x_{0b_1\cdots b_n}, y_{0b_1\cdots b_n})$.

We now proceed more formally.

Description of (P,V).

"$a = 4e$, where e is the constant of reduction CR. *Select* is P's random selector. $PAIR(n)$ is the set of pairs (x, y) such that $x \in BL(n)$ and $y \in NQR_x$."

Input to P and V:

- A random string σ, $\sigma = \rho\circ\tau_1\circ\tau_2$, where $|\rho| = 8n^3$, $|\tau_1| = 2n^a$ and $|\tau_2| = 2n^4$.
- A formula $\Phi \in 3SAT$ with n clauses.

Instructions for P:

P.1. "Choose and certify the first auxiliary pair."
Compute auxiliary pair $(x_0, y_0) = Select(\sigma, PAIR(n))$.
Send (x_0, y_0) and run algorithm A of §4 on input (x_0, y_0) and ρ. "Call $Proof_0$ the output."

P.2. "Choose and certify other auxiliary pairs."
Set $b_0 = 0$. Compute and send $b_0b_1b_2\cdots b_n$ =Select($\Phi, \{0,1\}^n$).
For $i = 0, \cdots, n$ do:

Set $s = b_0 \cdots b_i$.
Compute and send $(x_{s0}, y_{s0}) = Select(s0, PAIR(n))$ and $(x_{s1}, y_{s1}) = Select(s1, PAIR(n))$.
Compute $\Psi_s = CR(x_{s0}, y_{s0}, x_{s1}, y_{s1})$ and t_s, a satisfying assignment for Ψ_s.
Execute $\texttt{Prove}(\Psi_s, t_s, x_s, y_s, \tau_1)$. "Call $Proof\Psi_s$ the output."

P.3. "Prove $\Phi \in 3SAT$."
Set $s = b_0 \cdots b_n$. Let t_Φ be the lexicographically smaller satisfying assignment for Φ.
Execute $\texttt{Prove}(\Phi, t_\Phi, x_s, y_s, \tau_2)$. "Call $Proof\Phi$ the output."

Instructions for V:

"V receives from P the bits $b_0, b_1, \cdots, b_n$, (x_{b_0}, y_{b_0}), $(x_{b_0 0}, y_{b_0 0})$, $(x_{b_0 1}, y_{b_0 1}), \cdots$, $(x_{b_0 \cdots b_{n-1} 0}, y_{b_0 \cdots b_{n-1} 0})$, $(x_{b_0 \cdots b_{n-1} 1}, y_{b_0 \cdots b_{n-1} 1})$, the formulae $\Psi_{b_0}, \cdots, \Psi_{b_0 b_1 \cdots b_n}$, and the strings $Proof_0$, $Proof\Psi_{b_0}, \cdots, Proof\Psi_{b_0 \cdots b_n}$, $Proof\Phi$."

V.1. "Verify first auxiliary pair."
Run algorithm B of §4 on input ρ, (x_0, y_0), and $Proof_0$.
If B stops and rejects, stop and REJECT. Else,

V.2. "Verify other auxiliary pairs."
For $i = 1, \cdots, n$ do:
Set $s = b_0 \cdots b_i$.
Compute $\Psi_s = CR(x_{s0}, y_{s0}, x_{s1}, y_{s1})$.
If $\texttt{Check\_Prove}(\Psi_s, x_s, y_s, \tau_1, Proof\Psi_s)$=REJECT then stop and REJECT. Else,

V.3. "Verify $Proof\Phi$."
Compute n from $\rho \circ \tau_1 \circ \tau_2$ and verify that Φ has at most n clauses, and each of them has three literals. If not, stop and REJECT. Else,
Set $s = b_0 \cdots b_n$.
If $\texttt{Check\_Prove}(\Phi, x_s, y_s, \tau_2, Proof\Phi)$=REJECT then stop and REJECT. Else ACCEPT.

6.2. (P,V) is a noninteractive proof system for $3SAT$**.** The proof system (P, V) of §5 constitutes the main building block of the just-described sender–receiver pair (P, V). Therefore, the completeness of (P, V) can be easily derived from the analysis of completeness in §5.2.

Let us now focus our attention on the soundness. We shall show that, if the formula Φ is not 3-satisfiable, then for any Turing machine *Adversary* (even a "cheating" one that chooses Φ after seeing the reference string), V will accept the proof provided by *Adversary* with sufficiently low probability. The proof closely follows the reasoning done in §5.2 to prove the soundness of the proof system (P, V) described in §5.1. We distinguish two cases:

1. For some w, $(x_w, y_w) \notin \mathcal{NQR}(2n)$.
2. All the pairs (x_w, y_w) belong to $\mathcal{NQR}(2n)$ but $\Phi \notin 3SAT$.

If $(x_0, y_0) \notin \mathcal{NQR}(2n)$, we are in the very same situation analyzed in case (a) in the proof of soundness of §5.2. By the same reasoning, we conclude that the verification of step 1 is passed with sufficiently low probability. Suppose that for $w = sb$, where $b \in \{0, 1\}$, $(x_w, y_w) \notin \mathcal{NQR}(2n)$, and $(x_w, y_w) \in \mathcal{NQR}(2n)$. Then, $\Psi_w \notin 3SAT$ and therefore the procedure $\texttt{Check\_Prove}$ invoked for Ψ_w returns REJECT with sufficiently high probability.

Now, suppose that all pairs (x_w, y_w) belong to $\mathcal{NQR}(2n)$ but $\Phi \notin 3SAT$. Since $(x_s, y_s) \in \mathcal{NQR}(2n)$, $s = b_0 b_1 \cdots b_n$, following the reasoning done for cases (b) and (c) in the proof of soundness in §5.2, we conclude that verification step V.3 is passed with very low probability.

Now, we show that the proof system (P, V) is also zero-knowledge over $3SAT$.

6.3. The simulator. In this section, we describe an efficient algorithm S; in the next section we will prove that, on input of a sequence of 3-satisfiable formulae, S's output cannot, under the QRA, be distinguished from V's view by any efficient nonuniform algorithm.

S's Program

Input: An integer $n > 0$. A sequence $\Phi_1, \Phi_2, \cdots$ of 3-satisfiable formulae with n clauses each.

0. Set Sim_Output = empty string and $Tree$ = empty set.
1. "Choose ρ' and choose and certify first auxiliary pair."
   Randomly select two n-bit primes $p_0, q_0 \equiv 3 \bmod 4$ and set $x_0 = p_0 q_0$. Randomly select $y'_0 \in QR_{x_0}$.
   Execute procedure $\texttt{Gen\_}\rho\texttt{\_and\_Proof1}(x_0, y'_0)$, thus obtaining the strings ρ' and $Proof'_0$.
2. "Choose τ_1 and τ_2."
   Randomly select two strings τ_1 and τ_2 so that $|\tau_1| = 2n^a$ and $|\tau_2| = 2n^4$.
3. For each input formula Φ do:
   3.1. "Choose and certify other auxiliary pairs"
   Set $b_0 = 0$ and randomly select $b_1 \cdots b_n$. Append (x_0, y'_0), $Proof'_0$, and $b_0 b_1 \cdots b_n$ to Sim_Output. For $i = 0, \cdots, n$ do:
      Let $s = b_0 b_1 \cdots b_i$.
      If $s \notin Tree$ then
         Add s to $Tree$.
         Randomly select four n-bit primes $p_{s0}, q_{s0}, p_{s1}, q_{s1} \equiv 3 \bmod 4$.
         Set $x_{s0} = p_{s0} q_{s0}$ and $x_{s1} = p_{s1} q_{s1}$.
         Randomly select $y'_{s0} \in QR_{x_{s0}}$ and $y'_{s1} \in QR_{x_{s1}}$.
         Compute $\Psi_s = CR(x_{s0}, y'_{s0}, x_{s1}, y'_{s1})$.
         Execute procedure $\texttt{Gen\_Proof2}(\Psi_s, x_s, y'_s, p_s, q_s, \tau_1)$, thus obtaining $Proof\Psi'_s$.
      Append (x_{s0}, y'_{s0}), (x_{s1}, y'_{s1}), and $Proof\Psi'_s$ to Sim_Output.
   3.2. "Prove $\Phi \in 3SAT$."
   Set $s = b_0 b_1 \cdots b_n$. Execute $\texttt{Gen\_Proof2}(\Phi, x_s, y'_s, p_s, q_s, \tau_2)$ obtaining $Proof\Phi'$.
   Append $Proof\Phi'$ to Sim_Output.

Output: $(\rho' \circ \tau_1 \circ \tau_2, Sim\_Output)$

LEMMA 6.2. *Algorithm S is efficient.*

Proof. The running time of S is proportional to the number of input formulae. For each single input formula, all operations can be efficiently computed. Thus, S is efficient. (Note, again, that the running time is polynomial with respect to the input size, though it may be exponential in the parameter n.) □

The random variable output by S is certainly different from $View$ and, before proceeding any further, let us compare them. In $View$ the string ρ is truly random, while the corresponding string ρ' constructed by S does not contain any element in NQR_{x_0}. In $View$, each y_s is a quadratic nonresidue modulo the corresponding x_s, whereas in S, y'_s is chosen among the quadratic residues modulo x_s. Because of the different quadratic residuosity of the y_s's, the two distributions differ also in the Ψ_s's and in the strings $Proof\Psi_s$ and $Proof\Phi$. In fact, the formula Ψ_s is satisfiable if and only if both (x_{s0}, y_{s0}) and (x_{s1}, y_{s1}) are of the prescribed form. This is certainly the case in $View$. But in S, as all y_s's are quadratic residues, none of the pairs (x_s, y_s) is of the prescribed form and therefore none of the Ψ_s's is satisfiable. Moreover, the y_s's are also used to compute the labeling of the literals in the strings $Proof\Psi_s$'s and $Proof\Phi$'s and thus in S all literals are labeled with quadratic nonresidues.

In the next section, we shall prove, using a reasoning similar to the one in Section 5.3 that, despite the differences described above, the two families of random variables cannot be distinguished by any efficient nonuniform algorithm, under the QRA.

6.4. (P,V) is zero-knowledge.

THEOREM 6.3. *Under the* QRA, *the sender–receiver pair* (P, V) *of* §6.1 *is a noninteractive ZKPS.*

Proof. All that is left to prove is that (P, V) satisfies the zero-knowledge condition. We do this by showing that the output of algorithm S of the previous section cannot be distinguished from the view of the verifier V by any efficient nonuniform algorithm.

We proceed by contradiction. Assume that there exists a constant $d > 0$, an infinite subset $\mathcal{I} \subseteq \mathcal{N}$, a set $\{(\Phi_1^n, \Phi_2^n, \cdots)\}_{n\in\mathcal{I}}$ of sequences of 3-satisfiable formulae, where Φ_i^n has n clauses, and an efficient nonuniform algorithm $D = \{D_n\}_{n\in\mathcal{I}}$ such that for all $n \in \mathcal{I}$

$$\left|P_V(n) - P_S(n)\right| \geq n^{-d},$$

where

$$P_V(n) = Pr(s \xleftarrow{R} View(\Phi_1^n, \Phi_2^n, \cdots) \colon D_n(s) = 1)$$

and

$$P_S(n) = Pr(s \xleftarrow{R} S(1^n, \Phi_1^n, \Phi_2^n, \cdots) \colon D_n(s) = 1).$$

Let $R(n)$ be a polynomial such that the running time and the size of the program of each algorithm D_n is bounded by $R(n)$. Without loss of generality we can consider $R(n)$-tuples of 3-satisfiable formulae $\Phi_1^n, \cdots, \Phi_{R(n)}^n$, instead of arbitrary sequences of 3-satisfiable formulae $\Phi_1^n, \Phi_2^n, \cdots$.

As we have seen in the last section, the main difference between S's output and the view of the verifier is in the y_s's: they are all quadratic residues modulo the corresponding x_s's in S's output, while they are all quadratic nonresidues in $View$. We will now describe an efficient nonuniform algorithm $C = \{C_n\}_{n\in\mathcal{I}}$. Each C_n takes two inputs: $j \geq 0$ and $(x, y) \in PAIR(n) = \{(u, v) : u \in BL(n), v \in J_u^{+1}\}$; and has "wired-in" the formulae $\Phi_1^n, \cdots, \Phi_{R(n)}^n$ along with their lexicographically smaller satisfying assignments. Roughly speaking, C_n produces as output a "random" string and "proofs" for all formulae Φ_i^n's. C_n selects the input pair (x, y) as the jth auxiliary pair. All prior pairs are selected as simulator S does and all subsequent pairs as prover P does. Thus, C_n "knows" the factorization of the Blum modulus for all auxiliary pairs except (x, y). Nonetheless, algorithm C_n will use (x, y) as S would if $y \in QR_x$, and as P would if $y \in NQR_x$. More formally, C_n is designed so as to enjoy the following properties. Set

$$Space(n, j, QR) = \{x \xleftarrow{R} BL(n);\ y \xleftarrow{R} QR_x;\ s \xleftarrow{R} C_n(j, x, y) : s\},$$

$$Space(n, j, NQR) = \{x \xleftarrow{R} BL(n);\ y \xleftarrow{R} NQR_x;\ s \xleftarrow{R} C_n(j, x, y) : s\}.$$

Then,

Property (1) $Space(n, 0, NQR) = View(n, \Phi_1^n, \cdots, \Phi_{R(n)}^n)$,

Property (2) $Space(n, nR(n) + 1, QR) = \{s \xleftarrow{R} S(1^n, \Phi_1^n, \cdots, \Phi_{R(n)}^n) : s\}$,

Property (3) $Space(n, j, QR) = Space(n, j + 1, NQR)$.

From these properties we will conclude that the existence of D violates the QRA. We now formally describe the algorithm, and then prove all the stated properties.

The Algorithm C_n

"C_n has "wired-in" the $R(n)$-tuple $(\Phi_1^n, \cdots, \Phi_{R(n)}^n)$ and, for each $\Phi \in \{\Phi_1^n, \cdots, \Phi_{R(n)}^n\}$, the lexicographically smaller satisfying assignment t_Φ."

Input: "An integer $j \in [0, nR(n) + 1]$. A pair $(x, y) \in PAIR(n)$."

1. "Choose ρ and choose and certify first auxiliary pair."
   If $j = 0$ then set $x_0 = x$ and $y_0 = y$.
   Else randomly select two n-bit primes $p_0, q_0 \equiv 3 \bmod 4$, set $x_0 = p_0 q_0$, and select $y_0 \in QR_{x_0}$.
   Execute procedure `Gen_ρ_and_Proof1`(x_0, y_0), thus obtaining ρ and $Proof_0$.
2. "Choose other auxiliary pairs."
   "*Tree* contains the indices of auxiliary pairs that are used to certify two others auxiliary pairs. *Count* contains the number of all selected auxiliary pairs."
   Set $Tree$ = empty set and $Count = 1$.
   For each formula $\Phi \in \{\Phi_1^n, \cdots, \Phi_{R(n)}^n\}$ do:
      Set $b_0^\Phi = 0$ and randomly select n bits $b_1^\Phi, \cdots, b_n^\Phi$.
      For $i = 0, \cdots, n$ do:
         Set $s = b_0^\Phi \cdots b_i^\Phi$
         If $s \notin Tree$ then
            Add s to $Tree$. Randomly select four n-bit primes $p_{s0}, q_{s0}, p_{s1}, q_{s1} \equiv 3 \bmod 4$.
            "Choose 0-child."
            If $Count = j$ then set $x_{s0} = x$, $y_{s0} = y$.
            If $Count < j$ then set $x_{s0} = p_{s0} q_{s0}$ and randomly select $y_{s0} \in QR_{x_{s0}}$.
            If $Count > j$ then set $x_{s0} = p_{s0} q_{s0}$ and randomly select $y_{s0} \in NQR_{x_{s0}}$.
            $Count = Count + 1$
            "Choose 1-child."
            If $Count = j$ then set $x_{s1} = x$, $y_{s1} = y$.
            If $Count < j$ then set $x_{s1} = p_{s1} q_{s1}$ and randomly select $y_{s1} \in QR_{x_{s1}}$.
            If $Count > j$ then set $x_{s1} = p_{s1} q_{s1}$ and randomly select $y_{s1} \in NQR_{x_{s1}}$.
            $Count = Count + 1$
3. "Choose τ_1 and τ_2."
   Let w be the index of (x, y), that is $(x_w, y_w) = (x, y)$. If there is no such w, set w = empty string.[11]
   If $w \in Tree$ then

[11] It may happen that fewer than j (different) auxiliary pairs will be chosen. To give an extreme example, it may happen that, for all Φ, the bits $b_1^\Phi \cdots b_n^\Phi$ are always the same.

Compute $\Psi_w = CR(x_{w0}, y_{w0}, x_{w1}, y_{w1})$ and a satisfying assignment t_w for Ψ_w.
Execute procedure `Sample_τ_and_Proof2`(Ψ_w, t_w, x_w, y_w) obtaining τ_1 and $Proof\Psi_w$.
Randomly select a $2n^4$-bit string τ_2.
Else, if $w = b_0^\Phi \cdots b_n^\Phi$, for $\Phi \in \{\Phi_1^n, \cdots, \Phi_{R(n)}^n\}$, then
Execute procedure `Sample_τ_and_Proof2`(Φ, t_Φ, x, y) obtaining τ_2 and $Proof\Phi$.
Randomly select a $2n^a$-bit string τ_1.
Else, randomly select a $2n^a$-bit string τ_1 and a $2n^4$-bit string τ_2.
4. "Choose proofs with respect to τ_1 and τ_2."
Set $PROOF$= empty string and $Tree = \{w\}$.
For each formula $\Phi \in \{\Phi_1^n, \cdots, \Phi_{R(n)}^n\}$ do:
4.1. "Certify auxiliary pairs."
Append $(x_0, y_0), Proof_0$, and $b_0^\Phi \cdots b_n^\Phi$ to $PROOF$.
For $i = 0, \cdots, n$ do:
Set $s = b_0^\Phi \cdots b_i^\Phi$.
If $s \notin Tree$ then
Add s to $Tree$.
If $y_s \in NQR_{x_s}$ then
Compute $\Psi_s = CR(x_{s0}, y_{s0}, x_{s1}, y_{s1})$ and a satisfying assignment t_s for Ψ_s.
Execute procedure `Prove`$(\Psi_s, t_s, x_s, y_s, \tau_1)$ obtaining $Proof\Psi_s$.
If $y_s \in QR_{x_s}$ then execute `Gen_Proof2`$(\Psi_s, x_s, y_s, p_s, q_s, \tau_1)$ obtaining $Proof\Psi_s$.
Append (x_{s0}, y_{s0}), (x_{s1}, y_{s1}), and $Proof\Psi_s$ to $PROOF$.
4.2. "Prove Φ."
Set $s = b_0^\Phi \cdots b_n^\Phi$.
If $s \neq w$ then
If $y_s \in NQR_{x_s}$ then execute procedure `Prove`$(\Phi, t_\Phi, x_s, y_s, \tau_2)$ obtaining $Proof\Phi$.
If $y_s \in QR_{x_s}$ then execute `Gen_Proof2`$(\Phi, x_s, y_s, p_s, q_s, \tau_2)$ obtaining $Proof\Phi$.
Append $Proof\Phi$ to $PROOF$.

Output:$(\rho \circ \tau_1 \circ \tau_2, PROOF)$.

First note that $\{C_n\}_{n\in\mathcal{I}}$ is an efficient nonuniform algorithm. All x_s's (except the jth) are selected along with their prime factors and thus all related computations can be performed in expected polynomial time. All operations concerning x and y are simple multiplications and testing of membership in J_x^{+1}. The size of the set $Tree$ is never bigger than $nR(n)$, and thus membership and add operations are easily performed.

The strings τ_1 and τ_2 constructed by C_n are random. Indeed, either they are randomly selected or they are generated by `Sample_τ_Proof2`. The analysis in §5.4 shows that in the latter case the resulting string τ is random.

Proof of Property (1). Assume $j = 0$ and $y \in NQR_x$. All y_s's are quadratic nonresidues in C_n's output. (x, y) is set equal to (x_0, y_0) and used twice: at step 1 to produce ρ and $Proof_0$, and at step 3 to construct $Proof\Psi_0$. Both the strings $Proof_0$ and $Proof\Psi_0$ have the same probability of being chosen as in $View$ when the first pair is (x_0, y_0). From Lemma 5.3, each string ρ is equally likely to be constructed at step 1. Thus, $Space(n, 0, NQR) = View(n, \Phi_1^n, \cdots, \Phi_{R(n)}^n)$.

Proof of Property (2). Suppose $j = nR(n) + 1$. To prove $R(n)$ formulae, at most $nR(n)$ auxiliary pairs are needed. Thus, each y_s constructed by C_n belongs to QR_{x_s}. All the strings $Proof\Psi_s$'s and $Proof\Phi$'s are constructed in exactly the same way, both by S and by C_n. Hence, $Space(n, nR(n)+1, QR) = \{s \overset{R}{\leftarrow} S(1^n, \Phi_1^n, \cdots, \Phi_{R(n)}^n) : s\}$

Proof of Property (3). Consider now the two probability spaces $Space(n, j, QR)$ and $Space(n, j+1, NQR)$. In both spaces the auxiliary pairs are randomly chosen so that the first j y_s's are quadratic residues modulo the corresponding x_s's and, from the $(j+1)$st on, all the y_s's are quadratic nonresidues. All computations concerning pairs (x_s, y_s) different from (x, y) are performed in the same way. The pair (x, y) is used to construct either a proof $Proof\Psi_s$ for a formula Ψ_s derived from a reduction or a proof $Proof\Phi$ for one of the formulae Φ_i^n, or is never used. In the former two cases the proof is generated using the procedure `Sample_τ_and_Proof2`. When $y \in NQR_x$ ($y \in QR_x$), this procedure returns a string $Proof$ that has the same distribution as if it where generated by the procedure `Prove` (`Gen_Proof2`). Thus, $Space(n, j, QR) = Space(n, j+1, NQR)$.

We now conclude the proof of Theorem 6.3. We have assumed that D distinguishes between $S(1^n, \Phi_1^n, \cdots, \Phi_{R(n)}^n)$'s output and $View(n, \Phi_1^n, \cdots, \Phi_{R(n)}^n)$. From properties (1) and (2), then, this is tantamount to saying that D distinguishes between $Space(n, 0, NQR)$ and $Space(n, nR(n)+1, QR)$. By the pigeon-hole principle, and because of Property (3), for all $n \in \mathcal{I}$ there exists $j = j(n)$, $0 \leq j \leq nR(n)+1$, such that D distinguishes between $Space(n, j, QR)$ and $Space(n, j, NQR)$. That is, for all $n \in \mathcal{I}$,

$$\left|P_j(n, QR) - P_j(n, NQR)\right| \geq 1/((nR(n)+2)n^d)$$

where $P_j(n, QR) = Pr(s \overset{R}{\leftarrow} Space(n, j, QR) : D_n(s) = 1)$ and $P_j(n, NQR) = Pr(s \overset{R}{\leftarrow} Space(n, j, NQR) : D_n(s) = 1)$. Thus, composing each $C_n(j(n), \cdot, \cdot)$ with D_n, one obtains an efficient nonuniform algorithm that violates the QRA. □

6.5. Proving theorems of arbitrary size. Given a reference string of $8n^3 + 2n^a + 2n^4$ bit, the proof system (P, V) of §6.1 can be used to prove in zero-knowledge the 3-satisfiability of an arbitrary number of 3-satisfiable formulae, but each of them must have at most n clauses. However, the same proof system can be used to prove 3-satisfiable formulae with any number of clauses. The idea is perhaps best conveyed in an informal manner. Given a formula Φ with k clauses, the prover computes a certified auxiliary pair (x^Φ, y^Φ) and the lexicographically smaller satisfying assignment t for Φ. To label each literal u_j of Φ the prover randomly selects $r_j \in Z_{x^\Phi}^*$ and, if $t(u_j) = 1$ he associates with u_j the label $w_j = r_j^2 y^\Phi \bmod x^\Phi$; otherwise the label $w_j = r_j^2 \bmod x^\Phi$. The label associated with $\overline{u}_j$ is $w_j y^\Phi \bmod x^\Phi$. Essentially, a literal has an element in NQR_{x^Φ} as label if and only if it is made true by t. To prove that $\Phi \in 3SAT$, the prover proves that each clause has at least an element of NQR_{x^Φ} among the labels of its three literals. That is, consider the language $L = \{(y_1, y_2, y_3, x)$: at least one of y_1, y_2, y_3 belongs to $NQR_x\}$. Then $L \in NP$ and therefore there exists a fixed polynomial-time computable reduction RED such that

$$\Phi' = RED(y_1, y_2, y_3, x) \in 3SAT_{n^f} \iff (y_1, y_2, y_3, x) \in L,$$

where f is a fixed constant depending only on RED. Therefore, to prove that the ith clause is satisfied, the prover computes the formula Φ_i using the reduction RED and proves that $\Phi_i \in 3SAT$. By the property of the reduction the length of the formula is upper bounded by n^f and can thus be proved 3-satisfiable using the previously described proof system (P, V) with a reference string of $8n^{3f} + 2n^{sf} + 2n^{4f}$ bits. Therefore, we have reduced the problem of proving the 3-satisfiability of one formula with many clauses to that of proving the 3-satisfiability of many formulae, each with at most n^f clauses.

6.6. Efficient provers. In the proof system of §6.1, for convenience of presentation, the prover P was made quite powerful. For instance, P needs to find the lexicographically first satisfying assignment of a formula for proving that it is in 3SAT. This, however, is not necessary. It is easily seen that, under the QRA, the verifier would obtain an undistinguishable view [GoMiRa], no matter which satisfying assignment the prover may use. Also, it is possible for the prover to have access to a random oracle instead of a random selector and still generate essentially the same view to a polynomial-time verifier. In fact, by well-known techniques, a random oracle can be transformed to a random function associating each string with σ a "polynomially longer" random string. This random string may be used to select the necessary primes and quadratic residues and nonresidues with essentially the same odds as for a random selector. Actually, if one replaces a random oracle with a polyrandom function as in Goldreich, Goldwasser, and Micali [GoGoMi], the view of the verifier would still be indistinguishable from the one it obtains from P. These functions exist under the QRA[12] and the replacement only entails that the same short, randomly selected string should be remembered throughout the proving process.

In sum, the prover may very well be polynomial time, as long as it is given satisfying assignments for the formulae that need to be proved satisfiable in noninteractive zero knowledge.

This is an important point, and can be shown to hold not only for our specific noninteractive ZKPS, but also for any other that shares our algorithmic structure. Since, however, systems with a different structure and relying on weaker intractability assumptions have already been found (see below), we decline to formalize this point in our paper. Our goal, at this point, is making precise the notion of noninteractive zero-knowledge and showing its feasibility.

7. Recent improvements and related works. Two main open problems were posed in [DeMiPe1], namely,

1. whether many provers could share the same random string and[13]
2. whether it is possible to implement noninteractive zero-knowledge with a general complexity assumption, rather than on our specific number-theoretic one.

Recently, both our questions have been solved in a beautiful paper by Feige, Lapidot, and Shamir [FeLaSh]. They show that any number of provers can share the same random string and that any trap-door permutation can be used instead of quadratic residuosity. They also show that one-way permutations are sufficient for bounded noninteractive zero-knowledge, but the prover *needs* to have exponential computing

[12] In fact Blum, Blum, and Shub [BlBlSh] show that the QRA implies the existence of a polyrandom generator in the sense of Blum and Micali [BlMi] and Yao [Ya], and [GoGoMi] show that any polyrandom generator can be used to construct a polyrandom function.

[13] Indeed, if this had been done in our protocol, completeness and soundness would still hold. However, it is not clear that the zero-knowledge would be preserved. Without changing our proof systems, we can handle only a moderate number of provers. This number is limited for the same reasons outlined in footnote 6.

power. Our first question was also independently solved by De Santis and Yung [DeYu].

Noninteractive zero-knowledge has been shown to yield a new paradigm for digital signature schemes by Bellare and Goldwasser [BeGo].

De Santis, Micali, and Persiano [DeMiPe2] show that, if any one-way function exists, after an interactive preprocessing stage, any "sufficiently short" theorem can be proven noninteractively and in zero-knowledge.

Kilian, Micali, and Ostrovsky [KiMiOs] have shown that, if any one-way function exists, after a preprocessing stage consisting of a "few" executions of an oblivious transfer protocol, any theorem can be proven in zero knowledge and noninteractively. (Namely, after executing $O(k)$ oblivious transfers, the probability of accepting a false theorem is 1 in 2^k.) Bellare and Micali [BeMi] show that, based on a complexity assumption, it is possible to build public-key cryptosystems in which oblivious transfer is itself implementable without any interaction.

8. A general open problem. An obvious open problem in noninteractive zero-knowledge consists of finding more efficient proof systems. However, in our opinion, a more important one is decreasing the needed complexity assumption. This effort should be extended to all of cryptography at this point in its development.

Introducing new cryptographic primitives is crucial, but would be essentially impossible without first relying on some special, though hopefully well studied, complexity assumptions. It is important, though, to later find the minimal assumptions for implementing these primitives. In fact, "extra structure" may make proving that the desired property holds easier, but may also force the underlying complexity assumption to be false. Personally, Micali finds a dramatic difference between one-way functions and one-way permutations. (Breaking a glass is quite easy. Putting it back together is certainly harder, but what if we were guaranteed that there is a unique way to do so?)

We believe noninteractive zero-knowledge to be a fundamental primitive, one deserving the effort to establish the minimal assumptions needed for it to be securely implemented. We thus hope the following question will be settled: If one-way functions exist, does 3SAT have noninteractive zero-knowledge proof systems whose prover, given the proper witness, needs only to work in polynomial time?

9. Acknowledgments. We wholeheartedly thank Mihir Bellare for his constructive and generous criticism throughout this research.

Many thanks to Shafi Goldwasser and Mike Sipser for their encouraging support, technical and spiritual. (Make it also financial next time!)

Given that our work improves on that of [BlFeMi], we regret that we could not collaborate or reach Paul Feldman. We certainly would have benefited from his insights.

Finally, hoping that he or she accepts direct compliments, thanks to an anonymous referee for very intelligent comments.

REFERENCES

[AdHu] L. M. Adleman and M. A. Huang, *Recognizing primes in random polynomial time*, in Proc. 19th Annual ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 462–470.

[An] D. Angluin, *Lecture notes on the complexity of some problems in number theory*, Tech. Report 243, Yale University, Dept. of Computer Science, New Haven, CT, 1982.

[AnVa] D. ANGLUIN AND L. VALIANT, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

[Ba] L. BABAI, *Trading group theory for randomness*, in Proc. 17th Symposium on Theory of Computing, Providence, RI, 1985, pp. 421–429.

[BaMo] L. BABAI AND S. MORAN, *Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes*, J. Comput. System Sci., 36, (1988), pp. 254–276.

[BeGo] M. BELLARE AND S. GOLDWASSER, *New paradigm for digital signature and message authentication based on non-interactive zero-knowledge proofs*, in Advances in Cryptology–CRYPTO 89, Lecture Notes in Computer Science, 435, Springer–Verlag, Berlin, New York, 1989, pp. 194–211.

[BeGoWi] M. BEN-OR, S. GOLDWASSER, AND A. WIGDERSON, *Completeness theorems for non-cryptographic fault-tolerant distributed computations*, in Proc. 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 1–10.

[BeMi] M. BELLARE AND S. MICALI, *Non-interactive oblivious transfer and applications*, in Advances in Cryptology–CRYPTO 89, Lecture Notes in Computer Science, 435, Springer–Verlag, Berlin, New York, 1989, pp. 547–559.

[BeMiOs] M. BELLARE, S. MICALI, AND R. OSTROWSKY, *Perfect zero-knowledge in constant rounds*, in Proc. 22nd Annual ACM Symposium on the Theory of Computing, Baltimore, MD, 1990, pp. 482–493.

[BlBlSh] M. BLUM, L. BLUM, AND M. SHUB, *A simple and secure pseudo-random number generator*, SIAM J. Comput., 15 (1986), pp. 364–383.

[BlFeMi] M. BLUM, P. FELDMAN, AND S. MICALI, *Non-interactive zero-knowledge proof systems and applications*, in Proc. 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 103–112.

[BlMi] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequence of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[Bl1] M. BLUM, *Coin flipping by telephone*, IEEE COMPCON, High Technology in the Information Age, Spring 1982, pp. 133–137.

[Bl2] ———, *How to prove a theorem so no one else can claim it*, in Proc. Internat. Congress of Mathematicians, Berkeley, CA, 1986, pp. 1444–1451.

[BoHaZa] R. BOPPANA, J. HASTAD, AND S. ZACHOS, *Does co-NP have short interactive proofs?*, Inform. Process. Lett., 25 (1987), pp. 127–132.

[ChCrDa] D. CHAUM, C. CREPAU, AND I. DAMGÄRD, *Multiparty unconditionally secure protocols*, in Proc. 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 11–19.

[Co] S. A. COOK, *The complexity of theorem-proving procedures*, in Proc. 3rd Annual ACM Symposium on Theory of Computing, New York, NY, pp. 151–158.

[DeMiPe1] A. DE SANTIS, S. MICALI, AND G. PERSIANO, *Non-interactive zero-knowledge proof-systems*, in Advances in Cryptology–CRYPTO 87, Lecture Notes in Computer Science, 293 Springer–Verlag, Berlin, New York, 1987, pp. 52–72.

[DeMiPe2] A. DE SANTIS, S. MICALI, AND G. PERSIANO, *Non-interactive zero-knowledge proof-systems with preprocessing*, in Advances in Cryptology–CRYPTO 88, of Lecture Notes in Computer Science, 403, Springer–Verlag, Berlin, New York, 1988, pp. 269–283.

[DeYu] A. DE SANTIS AND M. YUNG, *Cryptograhic applications of the non-interactive metaproof and many-prover systems*, in Advances in Cryptology–Crypto 90, Springer–Verlag, Berlin, New York.

[ErSp] P. ERDOS AND J. SPENCER, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.

[FeFiSh] U. FEIGE, A. FIAT, AND A. SHAMIR, *Zero-knowledge proofs of identity*, in Proc. 19th Annual ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 210–217.

[FeLaSh] U. FEIGE, A. LAPIDOT, AND A. SHAMIR, *Multiple non-interactive zero-knowledge proofs based on a single random string*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 308–317.

[Fo] L. FORTNOW, *The complexity of perfect zero-knowledge*, in Proc. 19th Annual ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 204–209.

[FuGoMaSiZa] M. FURER, O. GOLDREICH, Y. MANSOUR, M. SIPSER, AND S. ZACHOS, *On completeness and soundness in interactive proof systems*, in Advances in Computing Research, Vol. 5. Randomness and Computation, S. Micali, ed., JAI Press Inc., Greenwich, CT, pp. 429–442.

[GaJo] M. GAREY AND D. JOHNSON, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, New York, 1979.

[GoGoMi] O. GOLDREICH, S. GOLDWASSER, AND S. MICALI, *How to construct random functions*, J. Assoc. Comput. Mach., 33 (1986), pp. 792–807.

[GoKi] S. GOLDWASSER AND J. KILIAN, *Almost all primes can be quickly certified*, in Proc. 18th Annual ACM Symposium on Theory of Computing, Berkeley, CA, 1986, pp. 316–329.

[GoMi1] S. GOLDWASSER AND S. MICALI, *Probabilistic Encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.

[GoMi2] ———, *Proofs with untrusted oracles*, manuscript.

[GoMiRa] S. GOLDWASSER, S. MICALI, AND C. RACKOFF, *The knowledge complexity of interactive proof-systems*, SIAM J. Comput., 18 (1989), pp. 186–208.

[GoMiRi] S. GOLDWASSER, S. MICALI, AND R. RIVEST, *A digital signature scheme secure against adaptive chosen-message attack*, SIAM J. Comput., 17 (1988), pp. 281–308.

[GoMiWi1] O. GOLDREICH, S. MICALI, AND A. WIGDERSON, *Proofs that yield nothing but their validity and a methodology of cryptographic design*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, Toronto, Ontario, Canada, 1986, pp. 174–187.

[GoMiWi2] ———, *How to play any mental game*, in Proc. 19th Annual ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 218–229.

[GoSi] S. GOLDWASSER AND M. SIPSER, *Private coins versus public coins in interactive proof-systems*, in Proc. 18th Symposium on Theory of Computing, Berkeley, CA, 1986, pp. 59–68.

[KaLi] R. KARP AND R. LIPTON, *Turing machines that take advice*, L'Enseignement Mathematique, 28, pp. 191–209, STOC 1980.

[KiMiOs] J. KILIAN, S. MICALI, AND R. OSTROWSKY, *Minimum resource zero-knowledge*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, Reserach Triangle Park, NC, 1989, pp. 474–479.

[MiSh] S. MICALI AND A. SHAMIR, *An improvement of the Fiat–Shamir identification and signature scheme*, Crypto 88, Lecture Notes in Computer Science 403, Springer–Verlag, Berlin, New York, 1988.

[NaYu] M. NAOR AND M. YUNG, *Public-key cryptosystems provably secure against chosen cyphertext attack*, in Proc. 22nd Symposium on Theory of Computing, Baltimore, MD, 1990, pp. 427–437.

[NiZu] I. NIVEN AND H. S. ZUCKERMAN, *An introduction to the theory of numbers*, John Wiley, New York, 1960.

[Ra1] M. RABIN, *Probabilistic algorithm for testing primality*, J. Number Theory, 12 (1980), pp. 128–138.

[Ra2] ———, *Digitalized signatures and public-key functions as intractable as factorization*, Tech. Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, Massachussetts Institute of Technology, Cambridge, MA, 1978.

[Ra3] ———, *Transaction protection by beacons*, Tech. Report 29-81, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1981.

[Sh] D. SHANKS, *Solved and unsolved problems in number theory*, Chelsea, New York, 1976.

[SoSt] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, SIAM J. Comput., 6 (1977), pp. 84–85

[Ya] A. YAO, *Theory and applications of trapdoor functions*, in Proc. 23rd IEEE Symposium on Foundations of Computer Science, Chicago, IL, 1982, pp. 80–91.