J. Cryptology (1991) 4: 185–206

Journal of Cryptology

Practical Zero-Knowledge Proofs: Giving Hints and Using Deficiencies[1]

Joan Boyar, Katalin Friedl, and Carsten Lund
Computer Science Department, University of Chicago,
Chicago, IL 60637, U.S.A.

Abstract. New zero-knowledge proofs are given for some number-theoretic problems. All of the problems are in NP, but the proofs given here are much more efficient than the previously known proofs. In addition, these proofs do not require the prover to be superpolynomial in power. A probabilistic polynomial-time prover with the appropriate trapdoor knowledge is sufficient. The proofs are perfect or statistical zero-knowledge in all cases except one.

Key words. Zero-knowledge proofs, Efficiency, Number-theoretic problems.

1. Introduction

Many researchers have studied zero-knowledge proofs and the classes of problems which have such zero-knowledge proofs. Little attention, however, has been paid to the practicality of these proofs. It is known, for example, that, under certain cryptographic assumptions, all problems in NP have zero-knowledge proofs [19], [8], [10]. Although these proofs can be performed with probabilistic polynomial-time provers who have the appropriate trapdoor information, these proofs may involve a transformation to a circuit or to an NP-complete problem, so they are often quite inefficient. The first zero-knowledge proofs, those for quadratic residuosity and nonresiduosity [22], were practical; they were efficient and the prover could be probabilistic polynomial-time if she[2] had the appropriate trapdoor knowledge. Other efficient zero-knowledge proofs are given in [9], [11], [12], [15], [23], and [30].

In this paper we present a practical zero-knowledge proof for a special case of primitivity. This protocol, which shows that an element of the multiplicative group modulo a prime is a generator, only requires that the prover be probabilistic polynomial time, though she must know the complete factorization of $p - 1$. Note that the protocol given in [30] is not practical because the prover must be able to

[1] Date received: January 7, 1989. Date revised: May 12, 1991. This research was supported in part by NSA Grant No. MDA904-88-H-2006.

[2] In this paper it will at times be convenient to think of the verifier as being named Vic, and the prover being named Peggy. Thus, "he" will refer to the verifier and "she" will refer to the prover.

compute discrete logarithms. In order to avoid that problem in our protocol, we have the verifier give the prover "hints" which will help her find the discrete logarithms in question.

Unfortunately, the portion of our protocol which shows that the element a is a primitive element of Z_p^* fails in some cases if $p - 1$ has large square factors. It fails, though, in such a well-defined manner that we can use its failure in a zero-knowledge proof that a number n is not square-free. This proof that a number is not square-free is zero-knowledge only under a certain reasonable intractability assumption and is thus only computational zero-knowledge rather than perfect or statistical zero-knowledge. The protocol does not, however, involve any bit encryptions (blobs). All previous "natural" zero-knowledge proofs which are neither perfect nor statistical zero-knowledge have used bit encryptions. Furthermore, this zero-knowledge proof is efficient, assuming the Extended Riemann Hypothesis.

We also give practical zero-knowledge proofs for nonprimitivity, and for membership and nonmembership in $\{n | n \text{ and } \varphi(n) \text{ are relatively prime}\}$. None of these proofs require that the prover be more than probabilistic polynomial time.

2. Definitions

This section contains definitions for interactive proofs and zero-knowledge [22].

Definition 1. An *interactive proof system* for a language L is a protocol for two probabilistic interactive Turing machines, the *prover* and the *verifier*. They have a common tape with the input string x. Both machines have private work tapes and private auxiliary input tapes, and there are two tapes on which they can communicate with each other. In polynomial time the verifier stops and either accepts or rejects the input string. The protocol has the following properties:

Completeness: if $x \in L$ and both the prover and the verifier are following the protocol, then, for every $c > 0$ $\Pr(\textit{verifier accepts } x) \geq 1 - |x|^{-c}$, for $|x|$ sufficiently large.

Soundness: if $x \notin L$ and the verifier is following the protocol, then, for every program run by the prover and for every $c > 0$, $\Pr(\textit{verifier rejects } x) \geq 1 - |x|^{-c}$, for $|x|$ sufficiently large.

Definition 2. An interactive proof system for a language L is *prover-practical* if the prover runs in probabilistic polynomial time. The prover's private auxiliary input tape is assumed to initially contain some trapdoor information about the input.

If P and V are the programs of the two interactive machines, then the interactive proof system is denoted by (P, V).

In the definition, the completeness property means that using the protocol the prover can convince the verifier of $x \in L$ with large probability. On the other hand, because of the soundness property, if $x \notin L$, the prover cannot convince the verifier of the contrary. The definition says that the probability that a cheating prover is successful should be less than $1/f(|x|)$ for any polynomial f. However, the protocols we present here follow the standard practice of only allowing an exponentially small probability of successful cheating.

In our paper we are interested in the case in which the running time of P is also polynomial in the length of the input, i.e., in prover-practical interactive proof systems. At the beginning of the protocol P has some additional information, "secret knowledge about the input," on her private auxiliary input tape. With this she can convince the verifier in polynomial time, that the input belongs to the language L.

Definition 3. A *transcript of a conversation* between machines V^* and P consists of the input string, the random bits of V^*, and the messages sent by the two parties.

In the following definitions we use Oren's notation [25]. The verifier may have some auxiliary input y on his private auxiliary input tape. In his definitions of zero-knowledge, Oren takes into account the effect that this auxiliary input has on the communication between the two parties. When these definitions are used, as opposed to the original definitions, the concatenation of two zero-knowledge protocols is still a zero-knowledge protocol.

Let $\langle P(x), V^*(x, y)\rangle$ denote the probability distribution of transcripts generated by P and V^* on $x \in L$, when y is initially on V^*'s private auxiliary input tape.

Intuitively, it is clear that if a machine M_{V^*}, which is no more powerful than the verifier, can produce transcripts which have a very similar distribution to $\langle P(x), V^*(x, y)\rangle$, then V^* will learn very little (other than that $x \in L$) which it could not have computed on its own. In order to formalize this idea of very similar transcripts, Goldwasser *et al.* [21] consider probabilistic polynomial-time distinguishers, which output 0 on some transcripts and 1 on others. If no distinguisher D can effectively differentiate between two distributions, they are considered similar.

Definition 4. An interactive proof system is *zero-knowledge* for the language L if, for every probabilistic polynomial-time machine V^*, there exists an expected polynomial-time algorithm M_{V^*}, such that, for every probabilistic polynomial-time machine D,

$$\forall c > 0, \quad \exists N > 0, \quad \forall x \in L, \quad \forall y,$$

$$|x| > N \quad \Rightarrow \quad |\Pr[D(\langle P(x), V^*(x, y)\rangle) = 0] - \Pr[D(M_{V^*}(x, y)) = 0]| \le \frac{1}{|x|^c}.$$

Note that $M_{V^*}(x, y)$ denotes the distribution of transcripts generated by M_{V^*}, given x and y as inputs.

M_{V^*}, the simulator, depends on the verifier's program V^*. For example, the simulator can use the verifier itself, run the verifier's program for a while, and occasionally back up the verifier's program to a certain point. Thus, we can think of the simulator as asking questions of the verifier (when it writes something on a communication tape and runs the program for the verifier to get a response), or as revealing information to the verifier (when it is responding to a challenge which the verifier's program has written on a communication tape). The simulator's output is a transcript.

In this general definition, the simulator's output is only *polynomially indistinguishable* from the original transcripts. The definitions below apply to certain cases in which it is possible to prove that the simulator's output is actually very similar to, rather than just polynomially indistinguishable from, the original transcripts. If the simulator's output has a distribution which is statistically very close to that of the original transcripts, we have statistical zero-knowledge; and if the distributions are identical, we have perfect zero-knowledge. We say that the protocol is *computational zero-knowledge* if it is zero-knowledge, but is not perfect or even statistical zero-knowledge.

Definition 5. An interactive proof system for the language L is *perfect zero-knowledge* if, for every probabilistic polynomial-time machine V^*, there exists an expected polynomial-time algorithm M_{V^*}, such that

$$\forall x \in L, \quad \forall y, \qquad \langle P(x), V^*(x, y)\rangle = M_{V^*}(x, y).$$

Definition 6. An interactive proof system for the language L is *statistical zero-knowledge* if, for every probabilistic polynomial-time machine V^*, there exists an expected polynomial-time algorithm M_{V^*}, such that, for any subset T of transcripts,

$$\forall c > 0, \quad \exists N, \quad \forall x \in L, \quad \forall y,$$

$$|x| > N \quad \Rightarrow \quad |\Pr[\langle P(x), V^*(x, y)\rangle \in T] - \Pr[M_{V^*}(x, y)) \in T]| \le \frac{1}{|x|^c}.$$

In practice, most statistical zero-knowledge proofs have also been perfect zero-knowledge proofs. Our imprimitivity protocol is an example of an interactive proof which is statistical, but not perfect, zero-knowledge.

It has been shown that if there exist any one-way functions, then every NP-language has a zero-knowledge proof system [19]. On the other hand, it is unlikely that there are perfect zero-knowledge proof systems for all problems with zero-knowledge proofs. The results of [17] and [7] show that NP-complete languages do not have perfect zero-knowledge proof systems unless the polynomial hierarchy collapses to the second level, which would be a major surprising result in complexity theory.

Zero-knowledge interactive proofs can be very useful in designing cryptographic protocols. If the subroutines in a cryptographic protocol are zero-knowledge, then they leak no information whatsoever, so it is easier to prove the entire protocol correct and secure. The tools which have been most useful in cryptography have been number theoretic, so we concentrate on proofs for number-theoretic problems.

Some of our proofs only work on a well-defined subset of the possible inputs, so these problems can be viewed as *promise problems* [14], [18]. From [14] we get the notation that a promise problem (Q, R) is deciding if the input x belongs to R given that we know that x belongs to Q.

The definitions of zero-knowledge proofs do not require that the prover be a probabilistic polynomial-time machine; hence zero-knowledge proofs may not be practical. None of our protocols require more than a probabilistic polynomial-time prover. Thus, they are prover-practical zero-knowledge proofs. In addition, none of our proofs involve a transformation to a circuit or an NP-complete problem.

Usually such a transformation would involve a significant blowup in the size of the problem, greatly increasing the number of bits which must be communicated. For example, the circuit for proving that the element g is a primitive element of Z_p^* would presumably involve checking that a factorization of $p-1$ is complete and checking, for each prime factor q of $p-1$, that g raised to the power $(p-1)/q$ is not the identity. This circuit is not at all trivial; the protocol we give involves much less communication.

We denote the number of bits communicated on inputs of size N, to achieve an error probability of no more than 2^{-N}, by $CC(N)$.

Showing that a protocol is a prover-practical zero-knowledge proof can be slightly more complicated than just showing that it is a zero-knowledge proof. When proving the completeness of the protocol, it is necessary to show that the prover can actually perform all of the required computations. This is, of course, unnecessary when no limits are placed on the prover's computational power.

3. The Zero-Knowledge Proofs

3.1. *Primitivity*

If we are allowing the prover to be all-powerful, it is easy to give a zero-knowledge proof that g is a generator of the multiplicative group modulo a prime p. In one such proof, the following would be repeated $k = \lceil \log_2 p \rceil$ times:

Protocol 1

1. The verifier randomly and uniformly chooses $r \in Z_{p-1}^*$.
2. The verifier computes $h \equiv g^r \pmod p$ and sends it to the prover.
3. The verifier and the prover execute Protocol 3 (see below). This will convince the prover that the verifier knows, in the sense of [15], the discrete logarithm of h.
4. The prover takes the discrete logarithm of h to get r.
5. The prover sends r back to the verifier who checks that it is correct.

This is slightly more complicated than the zero-knowledge proof in [30], and it still has the problem that the prover needs to be able to take discrete logarithms. If, instead of proving that g is a generator, we just want an interactive proof that g is a *quasi-generator*, then we do not need such a powerful prover. The verifier can give the prover a hint which enables her to compute the discrete logarithm. As we will see later, for many values of n, all quasi-generators will be generators.

Definition 7. Suppose p is a prime, g is a generator of Z_p^*, and $p-1 = 2^l p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$. Then any $g' = g^q \pmod p$ is called a *quasi-generator* if $q = p_1^{f_1} p_2^{f_2} \dots p_k^{f_k}$ where $0 \le f_i < e_i$ for all i.

In other words, g is a quasi-generator if and only if

(1) $g^{(p-1)/2} \neq 1$ and
(2) any odd prime dividing $p-1$ also divides the order of g.

Let us assume that the prover initially has the complete factorization of $p - 1$ on her private auxiliary input tape. In most applications, this is a reasonable assumption because it is possible in expected polynomial time to create a random prime p with a given length, along with the complete factorization of $p - 1$ [3], [1]. Now we modify the above zero-knowledge proof to include the following steps, which should be repeated $k = \lceil \log_2 p \rceil$ times:

Protocol 2

0. The verifier rejects if $g^{(p-1)/2} = 1$.
1. The verifier randomly and uniformly chooses $r \in Z_{p-1}^*$.
2. The verifier computes $h \equiv g^r \pmod p$ and sends it to the prover.

$2\frac{1}{2}$. The verifier computes $x \equiv r^2 \pmod{p-1}$ and sends it to the prover.

3. The verifier and prover execute Protocol 3 (see below). This will convince the prover that the verifier knows, in the sense of [15], the discrete logarithm of h.

4′. The prover takes the discrete logarithm of h to get r and checks that x has the correct form. If something fails, the prover terminates the protocol.

5. The prover sends r back to the verifier who checks that it is correct.

We now show that the above protocol is indeed a perfect zero-knowledge proof with a probabilistic polynomial-time prover.

Completeness. We show how a probabilistic polynomial-time prover can find r given the hint x and h. The idea is that the prover will solve the problem modulo every prime power dividing $p - 1$ and then use the Chinese Remainder Theorem to solve the problem modulo $p - 1$.

Let q be a prime dividing $p - 1$, and let l be maximal such that q^l divides $p - 1$. If $q = 2$, then x determines r modulo 2^l uniquely, since $r \equiv x^{2^{l-2}} \pmod{2^l}$.

If q is an odd prime, then x does not define r modulo q^l uniquely, since there are two square roots of any square in the ring Z_{q^l}. We show how the prover can find the one equal to r modulo q^l. The prover finds in polynomial time the two square roots r_1 and r_2 of x modulo q^l by using [2], [6], [26], or [27] to find the square roots modulo q and then lifting these solutions up to solutions modulo q^l. Without loss of generality, suppose $r_1 \equiv r \pmod{q^l}$ and $r_2 \equiv -r \pmod{q^l}$. Then there exist k_1 and k_2 such that $r_1 = r + k_1 q^l$ and $r_2 = -r + k_2 q^l$. Then

$$(g^{r_1} \cdot h^{-1})^{(p-1)/q^l} = (g^{r+k_1 q^l} \cdot g^{-r})^{(p-1)/q^l} = 1$$

and

$$\begin{aligned}(g^{r_2} \cdot h^{-1})^{(p-1)q^l} &= (g^{-r+k_2 q^l} \cdot g^{-r})^{(p-1)/q^l} \\ &= g^{-2r(p-1/q^l)} \neq 1\end{aligned}$$

since r was chosen from Z_{p-1}^* and g is a quasi-generator. Thus, the prover can simply compute $(g^{r_1} \cdot h^{-1})^{(p-1)/q^l}$ and $(g^{r_2} \cdot h^{-1})^{(p-1)/q^l}$ and choose the square root which produces the identity.

The prover calculates r modulo every prime power dividing $p - 1$ using the above procedure, and then she can calculate r using the Chinese Remainder Theorem.

Soundness. Let us suppose that g is not a quasi-generator. If g is a quadratic residue, the verifier rejects in step 0. Thus we may assume that g is a quadratic nonresidue. We show that in this case the prover fails to send back the correct r at least 50% of the time. If g is not a quasi-generator, then $g = f^{tq^l}$ for some $f \in Z_p^*$ and for some odd prime factor q of $p - 1$, such that q^l and $(p - 1)/q^l$ are relatively prime. Then there is another square root r' of x modulo $p - 1$ with $r' \equiv -r \pmod{q^l}$, but $r' \equiv r \pmod{(p-1)/q^l}$. This means that there exists an integer s such that $r' = r + s((p-1)/q^l)$ and $r' \not\equiv r \pmod{p-1}$. However, $g^{r'} = f^{tq^l(r+s((p-1)/q^l))} = f^{tq^l r} f^{ts(p-1)} = g^r$. Thus there are at least two distinct square roots of x which are discrete logarithms of h, so the prover cannot determine from x and h if the verifier chose r or r' in step 1.

Now we show that the prover learns nothing from the verifier in step 3 which could help her in determining which one of r and r' the verifier has chosen. Let us first describe the subprotocol used in step 3. This is the parallel version of the discrete logarithm protocol of [11] with the roles of the prover and the verifier switched. We are doing it in parallel to make it clearer that the entire primitivity protocol can be done in parallel.

The following is done in parallel for $1 \le i \le k = \lceil \log_2 p \rceil$.

Protocol 3

3.1. The verifier randomly and uniformly chooses $r_i \in Z_{p-1}$.
3.2. The verifier computes $h_i \equiv g^{r_i} \pmod p$ and sends it to the prover.
3.3. The prover chooses $\beta_i \in \{0, 1\}$ randomly and sends β_i to the verifier.
3.4. If $\beta_i = 0$, then the verifier sets $\hat{r}_i = r_i$; otherwise he sets $\hat{r}_i = r_i + r$. Then he reveals $\hat{r}_i$.
3.5. The prover checks that $h_i = g^{\hat{r}_i}/h^{\beta_i}$.

This protocol is in fact a witness hiding proof of knowledge [15], [16] of the discrete logarithm of h.

Look at the communication $(h, x, h_1, \ldots, h_k, \beta_1, \ldots, \beta_k, \hat{r}_1, \ldots, \hat{r}_k)$ at the point just before the prover reveals r. Recall that $r' = r + s((p-1)/q^l)$, that $r^2 = r'^2 \pmod{p-1}$, and that $h = g^r = g^{r'}$. Define

$$r_i' = \begin{cases} r_i & \text{if } \beta_i = 0, \\ r_i - s^{(p-1)/q^l} & \text{otherwise.} \end{cases}$$

The communication $(h, x, h_1, \ldots, h_k, \beta_1, \ldots, \beta_k, \hat{r}_1, \ldots, \hat{r}_k)$ arises in two equally likely situations, one in which Vic chose $(r, r_1, \ldots, r_k)$, and the other in which he chose $(r', r_1', \ldots, r_k')$ as his random choices. Observe that in both situations all of Vic's messages are the same, and these are the only possibilities, given that he chose either r or r' in step 1. Hence, this step is of no help to the prover, so she has at best a 50–50 chance of guessing whether the verifier chose r or r' in step 1.

Zero-Knowledge. We sketch some of the ideas for the construction of the simulator. The ideas follow the lines of [20]. The main idea is to use the verifier (here he can be any probabilistic polynomial-time machine), and his proof in step 3 that he knows r, to find this r.

The simulator asks a question $(\beta_1, \ldots, \beta_k)$ in step 3.3, and if it does not get a correct answer, meaning that, for all i, $(h_i, \hat{r}_i)$ satisfies that $h_i = g^{\hat{r}_i}/h^{\beta_i}$, it stops as the real prover would. If it gets a correct answer, it has to find the real r since this is what the real prover does. To do this, it resets the verifier to the point just before the question was asked and asks another random question $(\beta'_1, \beta'_2, \ldots, \beta'_k)$. If it gets a correct answer to $(\beta'_1, \beta'_2, \ldots, \beta'_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k) \neq (\beta'_1, \beta'_2, \ldots, \beta'_k)$, then it can find r, since if $\beta'_i \neq \beta_i$, we have $r = \pm(\hat{r}'_i - \hat{r}_i)$. If it cannot find r this way, then the simulator continues asking random questions until it can either find r or it has asked 2^k questions. In the second case, it computes r, using brute force. It can be shown that this simulator runs in expected polynomial time for all verifiers. Furthermore, we can make this a bounded round protocol because this simulator works even if the protocol is run in parallel. In Appendix A we given the details of this simulation. Hence we get a bounded round perfect zero-knowledge protocol.

Since $O(k \log_2 p)$ bits are communicated in step 3 to achieve error probability not greater than $(\frac{1}{2})^k$, the communication cost of the entire protocol is $O(k^2 \log_2 p)$. This gives

Theorem 1. *Let*

$$L = \{(g, p) | p \text{ prime and } g \text{ is a quasi-generator of } Z_p^*\}.$$

Then there is a prover-practical perfect zero-knowledge, bounded round, interactive proof system for L.

The prover's auxiliary input tape contains the complete factorization of $p - 1$. *The communication cost of this protocol is* $CC(N) = O(N^3)$, *where* $N = 2\lceil \log_2 p \rceil$ *is the size of the input.*

If $p - 1$ is square-free, then a quasi-generator is in fact a generator. This fact gives the following corollary.

Corollary 1. *Let*

$$Q = \{(g, p) | p \text{ prime and } p - 1 \text{ is square-free}\}$$

and

$$G = \{(g, p) | p \text{ prime and } g \text{ is a generator of } Z_p^*\}.$$

Then there is a prover-practical perfect zero-knowledge, bounded round, interactive proof system for the promise problem (Q, G).

The prover's auxiliary input tape contains the complete factorization of $p - 1$. *The communication cost of this protocol is* $CC(N) = O(N^3)$, *where* $N = 2\lceil \log_2 p \rceil$ *is the size of the input.*

The set of primes, for which this protocol can be used to "prove" that an element is a generator, is of reasonable size since [29] proved that

$$\exists c > 0, \qquad \frac{\{p | p \leq x, p \text{ prime and } p - 1 \text{ square-free}\}}{\{p | p \leq x \text{ and } p \text{ prime}\}} \geq c$$

for x sufficiently large.

Throughout this section we have been looking at the multiplicative group Z_p^* of the integers modulo a prime p. It is easy, however, to generalize the proof system given above to any other cyclic group with known order. For the proof that an element is a generator it is enough to assume that the order of the group is square-free except for some "easy-to-find" prime divisors with exponents larger than 1. (Known prime divisors can be handled as 2 is in step 0.) Consider, for example, the multiplicative group Z_q^* of the integers modulo $q = p^n$, where p is an odd prime and $n \geq 1$. Almost all that is necessary is to substitute $\varphi(q) = p^{n-1}(p-1)$ in place of $p - 1$ throughout this exposition. (When q is prime, $\varphi(q)$, Euler's phi function, has the value $q - 1$.) Of course, $\varphi(q)$ is never square-free if $n > 2$. This is not a problem, however, because q is easy to factor, so the verifier and the simulator can find p and can check that $g^{p^{n-2}(p-1)} \not\equiv 1 \pmod{p^n}$. Thus, we can assume that $g \not\equiv h^{tp}$ (mod p^n) for any integer t, and we again only need to worry about square factors of $p - 1$.[3]

3.2. *Are n and $\varphi(n)$ Relatively Prime?*

In the zero-knowledge proof system for generators presented in the previous section, we had to assume that $p - 1$ was square-free. This is unfortunate, particularly since there is no known efficient zero-knowledge proof for square-freeness. It is possible, however, to give an efficient proof that a number n and $\varphi(n)$, the number of elements in the multiplicative group modulo n, are relatively prime. This property implies that n is square-free. Thus, if $p - 1 = 2^l m$, where m is odd, and if m and $\varphi(m)$ are relatively prime, the prover could prove that this is the case and afterward she could prove primitivity. Unfortunately, it is possible to have m and $\varphi(m)$ not relatively prime even if $p - 1$ is square-free, so this proof system will not work for quite as large a class as we would like. Combined with the proof system of the previous section, however, it gives a perfect zero-knowledge proof for

$$\{(p, g) \mid p \text{ is prime}, p - 1 = 2^l m, \text{ where } m \text{ is odd}, \gcd(m, \varphi(m)) = 1, \text{ and } \langle g \rangle = Z_p^*\}.$$

Suppose the prover knows $\varphi(n)$ for an odd integer n and wants to prove that n and $\varphi(n)$ are relatively prime. The prover and verifier can repeat the following $\lceil \log_2 n \rceil$ times.

Protocol 4

1. The verifier randomly and uniformly chooses $x \in Z_n^*$ and sends it to the prover.
2. The prover chooses a random $r \in Z_n^*$ and sends the verifier $y \equiv r^n x \pmod{n}$.
3. The verifier chooses $\beta \in \{0, 1\}$ randomly with equal probabilities and sends β to the prover.
4. If $\beta = 0$, the prover reveals r showing that y was formed correctly. If $\beta = 1$, the prover reveals an nth root of y, thus showing that x has an nth root modulo n.

---

[3] Notice that this is even easier in this particular case because the problem of determining primitivity in the group $Z_{p^n}^*$ is efficiently reducible to that of determining primitivity in Z_p^*. This follows from the fact that an element $g \in Z_{p^n}^*$ is primitive if and only if $g^{p^{n-2}(p-1)} \not\equiv 1 \pmod{p^n}$ and g is primitive when viewed as an element of the group Z_p^*.

We now show that the above is a perfect zero-knowledge interactive proof system for $\{n | \gcd(n, \varphi(n)) = 1\}$.

Completeness. When n and $\varphi(n)$ are relatively prime, $x \equiv (x^k)^n \pmod{n}$ where $k \equiv (n \pmod{\varphi(n)})^{-1} \pmod{\varphi(n)}$. Hence the prover can compute nth roots of x and y.

Soundness. Suppose that n and $\varphi(n)$ are not relatively prime. Then the $\gcd(n, \varphi(n)) = q$, where $1 < q < \varphi(n) < n$. Since there is some positive integer t such that, for every $g \in Z_n^*$, $g^n \equiv g^{tq} \pmod{n}$, every element which has nth roots also has qth roots. Exactly $\varphi(n)/q$ elements in Z_n^* have qth roots, so no more than half of the elements of Z_n^* have nth roots. If the verifier chooses an x which does not have an nth root, there is no more than a 50–50 chance that the prover will be able to answer the challenge chosen by the verifier. Thus, at each step, there is at least one chance in four that the prover will be caught, making the probability that the prover will succeed $\lceil \log_2 n \rceil$ times exponentially small.

Zero-Knowledge. The simulator gets x from the verifier and chooses randomly and uniformly $\gamma \in \{0, 1\}$ and $r \in Z_n^*$. If $\gamma = 0$, it lets $y = r^n x$; otherwise $y = r^n$. Observe that since x has an nth root, we have that y in both cases is drawn from the same distribution as that of the prover's y. So there is a 50–50 chance that the verifier will choose $\beta = \gamma$. In this case the simulation succeeds; otherwise the simulator backs up the verifier, chooses new random γ and r, and tries again. Thus the simulation is expected polynomial time, and this protocol is perfect zero-knowledge.

Furthermore, the protocol can be parallelized following the lines of [4], as Protocol 5 below is parallelized in Protocol 6, giving a bounded round, perfect zero-knowledge proof system. The above discussion gives

Theorem 2. *There is a prover-practical perfect zero-knowledge, bounded round, interactive proof system for*

$$\{n | \gcd(n, \varphi(n)) = 1\}$$

with communication cost $CC(N) = O(N^2)$, where $N = \lceil \log_2 n \rceil$ is the size of the input. The prover's auxiliary input tape contains the number $\varphi(n)$.

Corollary 2. *There is a prover-practical perfect zero-knowledge, bounded round, interactive proof system for*

$$\{(p, g) | p \textit{ prime}, p - 1 = 2^l m, \textit{where } m \textit{ is odd}, \gcd(m, \varphi(m)) = 1, \textit{and } \langle g \rangle = Z_p^*\}.$$

The prover's auxiliary input tape contains the complete factorization of $p - 1$. The communication cost is $CC(N) = O(N^3)$, where $N = 2\lceil \log_2 p \rceil$ is the size of the input.

If n and $\varphi(n)$ are not relatively prime, a prover who knows $\varphi(n)$ can give a prover-practical zero-knowledge proof that they have a common factor, under certain assumptions. One such proof involves repeating the following $\lceil \log_2 n \rceil$ times. First, the prover sends the verifier a random $x \in Z_n^*$ such that x does not have an nth root. She can do this by choosing random $x \in Z_n^*$ until $x^{\varphi(n)/\gcd(n, \varphi(n))} \not\equiv 1 \pmod{n}$.

Then, the verifier chooses a random $r \in Z_n^*$ and a random bit β. The verifier then sends $y \equiv r^n x^\beta \pmod{n}$ to the prover. Next, using the technique due to Benaloh [5] of using cryptographic capsules, the verifier gives a zero-knowledge proof that he knows n and β. Finally, the prover reveals the bit β. The reason this is not perfect zero-knowledge is that the prover must originally produce an nth-nonresidue x, and it is not clear that the simulator can do this. If $q = \gcd(n, \varphi(n))$ is large enough (superpolynomial) though, the simulator could pick $x \in Z_n^*$ at random and it is unlikely that x would be a qth-residue. In this case, the protocol would be statistical zero-knowledge.

3.3. *Imprimitivity*

Suppose p is a prime and g is not a generator of Z_p^*. In this section we show how, if the prover knows $t < p - 1$ such that $g^t \pmod{p} \equiv 1$, she can give a prover-practical interactive proof that g is not a generator. The proof is statistical zero-knowledge if $(p - 1)/t$ is large enough. The major advantage of the protocol given here over that in [30] is that we do not need to assume that a generator for Z_p^* is publicly available. The set we are concerned with is

$$S = \{(p, g) | p \text{ is a prime}, \exists t < p - 1, g^t \equiv 1 \pmod{p}\}.$$

The values p and g are available to both the prover and the verifier; the value t is initially on the prover's private auxiliary input tape; and the prover is attempting to convince the verifier that g is not a generator modulo p. Let $s = (p - 1)/t$. Our proof is based on the fact that for every integer r, l, $g^r \equiv g^{r+tl} \pmod{p}$, so the prover can find many discrete logarithms for an element as long as she knows one discrete logarithm. If g was a generator, however, each element would have only one discrete logarithm in the range $[1, p - 1]$. The protocol consists of $\lceil \log_2 p \rceil$ independent repetitions of the following:

Protocol 5

1. The prover chooses a random r uniformly from the range $[1, t]$.
2. The prover sends the verifier $h \equiv g^r \pmod{p}$.
3. The verifier chooses $\beta \in \{0, 1\}$ randomly with equal probabilities and sends β to the prover.
4. If $\beta = 0$, the prover chooses a random z uniformly from $[0, \lfloor s/2 \rfloor - 1]$. If $\beta = 1$, the prover chooses a random z uniformly from $[\lceil s/2 \rceil, s - 1]$.
5. The prover sends the verifier $r' = r + zt$ who checks that $h \equiv g^{r'} \pmod{p}$ and that $r' \in [1, (p - 1)/2]$ if $\beta = 0$, or that $r' \in [(p - 1)/2 + 1, p - 1]$ otherwise.

Completeness. Notice that in step 5 the prover is revealing a discrete logarithm of h which is less than $(p - 1)/2$ if the verifier's a challenge was $\beta = 0$, or greater than $(p - 1)/2$ if $\beta = 1$. If g is not a generator, for all $h \in \langle g \rangle$, two such discrete logarithms will exist, and the method described for computing them is efficient.

Soundness. If g was a generator, only one discrete logarithm would exist, so for each of the verifier's challenges, the prover would have at most a 50–50 chance of being able to give a correct response.

Zero-Knowledge. Let us look at a simulator for this protocol. The simulator would choose a random r uniformly from $[1, p - 1]$. The simulator would then run the program for the verifier with the value g^r being sent from the prover. The simulator has a 50–50 chance of answering the verifier's question each time simply by revealing r. If it canot answer, it will backtrack the verifier to the point of choosing r and try another one. The simulation is obviously expected polynomial time. Both the prover and the simulator choose h to be a random element of the subgroup generated by g. If s is even the simulator generates r''s with the same distribution as the prover. The interesting case is when s is odd (because otherwise g is a quadratic residue) and then the distributions of r''s in step 5 are somewhat different depending on whether you have the true prover or the simulator.

The true prover never gives r' in the interval

$$\left[\frac{p-1}{2} - \frac{t}{2} + 1, \frac{p-1}{2} + \frac{t}{2}\right]$$

if s is odd, but the simulator might. But since s is large, these distributions are statistically close. Let us look at one of the independent repetitions of the above protocol. Let $P(x)$ denote the probability that the true prover reveals x in step 5, and let $S(x)$ denote the probability that the simulator produces x in step 5. For any subset X of $\{1, \ldots, p - 1\}$, $|\sum_{x \in X} P(x) - \sum_{x \in X} S(x)| \leq 1/s$. Hence for the whole protocol the distributions differ by at most $(\log_2 p)/s$. Thus this protocol is statistical zero-knowledge for the languages which are subsets of S of the form

$$S_f = \left\{(p, g) | p \text{ prime}, \exists t < p - 1, g^t \equiv 1 \pmod{p} \text{ and } \frac{p-1}{t} \geq f(\log p)\right\},$$

where f is any *superpolynomial* function (i.e., which grows faster than any polynomial).

The communication cost of this protocol is $CC(N) = O(N^2)$, where $N = 2\lceil \log_2 p \rceil$ is size of the input. The above discussion gives

Theorem 3. *There is a prover-practical interactive proof system for*

$$\{(p, g) | p \textit{ is a prime}, \exists t < p - 1, g^t \equiv 1 \pmod{p}\}$$

and it is statistical zero-knowledge on

$$S_f = \left\{(p, g) | p \textit{ is a prime}, \exists t < p - 1, g^t \equiv 1 \pmod{p} \textit{ and } \frac{p-1}{t} \geq f(\log_2 p)\right\},$$

where f is superpolynomial. This protocol has communication cost $CC(N) = O(N^2)$, where $N = 2\lceil \log_2 p \rceil$ is the size of the input. The prover's auxiliary input tape contains t.

This restriction to subsets S_f of S is unfortunate. If the prover only proves things from these smaller sets, she gives away some information, i.e., that $s \geq f(\log_2 p)$. This does not appear to be much information since if s is small the verifier could himself have found s. But since there is a grey area between superpolynomially large and any fixed polynomial, we cannot find a uniform simulator that works for all possible magnitudes for s. One solution to this problem is to consider an alternative definition of zero-knowledge. In the GMR-definition we have a simulator which can fool every probabilistic polynomial-time distinguisher with probability greater than $1 - 1/N^c$ for every c for N sufficiently large, where N is the input size. In our definition we give c to the simulator, which then runs in an expected time which is polynomial in N^c. Hence the simulator is expected polynomial time for fixed c. Other than allowing the simulator's running time to vary depending on c, this definition is identical to Oren's [25], and we are using similar notation.

Definition 8. Let (P, V) be a interactive proof system for L. Then (P, V) is *weak zero-knowledge* if, for every probabilistic polynomial-time machine V^*, there exists an algorithm $M_{V^*}(c, x, y)$ which runs in expected polynomial time for fixed c, such that, for every probabilistic polynomial-time machine D,

$$\forall c, \quad \exists N, \quad \forall x \in L, \quad \forall y,$$

$$|x| > N \Rightarrow |\Pr[D(\langle P(x), V^*(x, y)\rangle) = 0] - \Pr[D(M_{V^*}(c, x, y)) = 0]| \leq \frac{1}{|x|^c}.$$

It is *weak statistical zero-knowledge* if, for every probabilistic polynomial-time machine V^*, there exists an algorithm $M_{V^*}(c, x, y)$ which runs in expected polynomial time for fixed c, such that, for any subset T of transcripts,

$$\forall c, \quad \exists N, \quad \forall x \in L, \quad \forall y,$$

$$|x| > N \quad \Rightarrow \quad |\Pr[D(\langle P(x), V^*(x, y)\rangle \in T] - \Pr[M_{V^*}(c, x, y)) \in T]| \leq \frac{1}{|x|^c}.$$

We believe that this definition captures the intuition of zero-knowledge.

With this definition we can easily construct a simulator for the nongenerator protocol. It behaves exactly as the old one after testing that $s \geq \log^{c+2} p$. If it finds s and hence t, it proceeds as the real prover would; otherwise it proceeds as the old simulator would. Using our new definition we get:

Theorem 4. *There is a prover-practical weak statistical zero-knowledge interactive proof system for*

$$\{(p, g) | p \text{ is a prime}, \exists t < p - 1, g^t \equiv 1 \pmod{p}\}$$

with communication cost $CC(N) = O(N^2)$, *where* $N = 2\lceil \log_2 p \rceil$ *is the size of the input. The prover's auxiliary input tape contains* t.

With this new definition of zero-knowledge we can also remove the assumption, in the protocol in [30] for the same problem, that one generator is publicly known. We can let the prover give the verifier a random generator. This is prover-practical weak zero-knowledge because the simulator can find a generator with probability $1 - \log^{-c} n$ in time polynomial in $\log^c n$, as shown in Appendix B.

The proof system presented above can be extended to work for many other cyclic groups with known order. In particular, when working with the multiplicative group modulo q, a power of an odd prime p, all that is necessary is to substitute $\varphi(q) = p^{n-1}(p - 1)$ in place of $p - 1$ throughout this exposition.

Furthermore, the protocol can be parallelized using techniques similar to those of [4]. Let $k = \lceil \log p \rceil$.

Protocol 6

1. The prover chooses randomly and uniformly $x \in Z_{p-1}^*$, computes $f \equiv g^x \pmod p$, and sends f to the verifier.
2. The verifier chooses randomly and uniformly all his challenges $(\beta_1, \ldots, \beta_k)$ and commits to them by choosing a random $s_i \in Z_{p-1}^*$ for each β_i. If $\beta_i = 0$, he lets $t_i \equiv g^{s_i} \pmod p$; otherwise $t_i \equiv f^{s_i} \pmod p$. He sends $(t_1, t_2, \ldots, t_k)$ to the prover.
3. For $1 \le i \le k$, the prover chooses a random r_i uniformly from the range $[1, t]$.
4. The prover sends the verifier $(h_1, \ldots, h_k)$, where $h_i \equiv g^{r_i} \pmod p$.
5. The verifier reveals his challenges by sending $(\beta_1, \ldots, \beta_k)$ and $(s_1, s_2, \ldots, s_k)$.
6. The prover checks that $(g^{x^{\beta_i}})^{s_i} = t_i$ for $1 \le i \le k$.
7. If $\beta_i = 0$, the prover chooses a random z_i uniformly from $[0, \lfloor s/2 \rfloor - 1]$. If $\beta_i = 1$, the prover chooses a random z_i uniformly from $[\lceil s/2 \rceil, s - 1]$.
8. The prover sends $(r_1', \ldots, r_k')$, where $r_i' = r_i + z_i t$.
9. The verifier checks that $h_i \equiv g^{r_i'} \pmod p$ and that $r_i' \in [1, (p-1)/2]$ if $\beta_i = 0$, or that $r_i' \in [(p-1)/2 + 1, p - 1]$ otherwise. Furthermore, he checks that $f = g^x$.

Completeness. The completeness is obvious from the completeness of the non-parallel version of the protocol.

Soundness. To see that this soundness is preserved, observe that the two different commitments come from the same distribution since $x \in Z_{p-1}^*$. Thus receiving these commitments earlier is of no help to Peggy.

Zero-Knowledge. The simulator is constructed as follows. It does the same as Peggy until Vic reveals all his challenges. Then it backtracks to the point where Vic has just made his commitments. Now the simulator forms new h_i's so that it can answer Vic's questions. (This is the set of h_i's that it will output as part of the transcript.) If Vic reveals the same old questions, then the simulator can answer them. Otherwise the simulator learns s, s' such that $g^s = f^{s'}$. This gives

$$g^s = g^{xs'} \quad \Rightarrow \quad g^{s-xs'} = 1.$$

It is easy to see that, since the simulator chooses x randomly, $s - xs' \pmod{p-1}$ is a random multiple at of t, the order of g. Now the simulator will repeat the above procedure until it succeeds in getting another random multiple $a't$ or until it has run the procedure 2^k times, in which case it will find t by brute force. We know from [24] that $\Pr[\gcd(at, a't) = t] = 6/\pi^2$. Hence, it can be shown, by techniques similar to those in Appendix A, that this simulator runs in expected polynomial time.

If the modulus has more than one prime factor or is a large power (≥ 3) of two, no elements would be generators. We could, however, still ask the question: Does the subgroup generated by the element g have fewer than m elements (for a prime modulus m can be $p - 1$)? Then if the prover knows t such that $g^t \equiv 1 \pmod{n}$, and $s = \lfloor m/t \rfloor$ is sufficiently large, we could give a zero-knowledge proof that g only generates a small subgroup.

3.4. *Does n Have a Square Factor?*

Recall that in Section 3.1 (Corollary 1) we constructed a protocol for proving that an element is a generator for Z_p^*, where p is a prime and $p - 1$ is square-free. When $p - 1$ is not square-free, that protocol shows that an element is a quasi-generator, though it may not be a generator. It is possible, however, to use this deficiency in the proof system for primitivity to show that $p - 1$ is not square-free.

First, if $4 | p - 1$, $p - 1$ clearly has a square factor and this is easily seen by the verifier, so we can assume that $4 \nmid p - 1$. Therefore if $p - 1$ is not square-free, then the prover can find an element h that is a quasi-generator but not a generator. Now she can prove to the verifier that h is a quasi-generator using Protocol 2 and that h is not a generator using Protocol 5. At this point the verifier is convinced that $p - 1$ has a square factor.

This idea can be extended to give a general protocol for integers which have a square factor. To show that an integer n has a square factor, the prover first finds a prime p such that $n | p - 1$. Then the prover shows that $p - 1$ has a square factor, which is also a square factor of n.

The set we are concerned with is

$$S = \{n | n = q^2 m,\ q \text{ prime}\}.$$

The integer n is available to both the prover and the verifier; the complete factorization of n is initially on the prover's private auxiliary input tape; and the prover is attempting to convince the verifier that n has a nontrivial square factor, q^2, where q is prime.

Protocol 7

1. The verifier accepts if $4 | n$.
2. The prover finds a prime $p < n^3$ such that $p = an + 1$. She sends p and the complete factorization of a to the verifier.
3. The verifier accepts if $s^2 | n$ for some prime factor s of a.
4. The prover finds a generator g in Z_p^* and sends $h = g^q$ to the verifier.
5. The prover proves that h is a quasi-generator, using Protocol 2.
6. The prover proves that h is not a generator, using Protocol 5.
7. The verifier checks that $h^{(p-1)/s} \neq 1$ for all prime factors s of a.

Completeness. Assuming the Extended Riemann Hypothesis, we can try random a's which are less than n^2 and expect to find p in $O(\log n)$ attempts. To see this, consider the following (from pp. 129 and 136 of [13]). Assuming the Extended Riemann Hypothesis,

$$|\{p | p \text{ prime},\ p \leq x,\ p \equiv 1 \pmod{n}\}| = \frac{\operatorname{li} x}{\varphi(n)} + O(x^{1/2} \log x),$$

where

$$\operatorname{li} x = \int_2^x \frac{1}{\log t}\,dt = \frac{x}{\log x} - \frac{2}{\log 2} + \int_2^x \frac{1}{\log^2 t}\,dt > \frac{x}{\log x} + O(1).$$

Hence the probability that a random m, chosen so that $m \equiv 1 \pmod n$ and $m \leq x$, is prime is

$$\frac{x/(\varphi(n)\log x) + O(1)/\varphi(n) + O(x^{1/2}\log x)}{\lfloor (x-1)/n \rfloor}.$$

We have from [28] that $\varphi(n) \geq C(n/\log\log n)$; hence if $x = n^3$ the above is greater than

$$C'\frac{\log\log n}{\log n} + O(n^{-1/2}\log n).$$

Note that $x = n^{2+\varepsilon}$ is sufficient if $\varepsilon > 0$.

To find p, we can use Bach's method [3] to produce an appropriate a randomly, along with the complete factorization of a.

Another way to find an appropriate p is by trying $n+1, 2n+1, 3n+1, \ldots$ until we find a prime. Wagstaff [31] has given a heuristic argument which says that we would usually only have to try up to $O(\log^2 n)$ numbers. Observe that we can factor a since it is so small.

Because of step 3, we can assume that $q \nmid a$, so the prover will succeed in showing both that h is a quasi-generator and that it is not a generator.

Soundness. Assume that n is square-free. As observed before, if $p-1$ is square-free there will be no quasi-generators which are not generators. So it is very unlikely that the verifier will accept the prover's proofs in step 5 and step 6 unless $p-1$ has a factor s such that $s^2 \mid (p-1)$ and $h^{(p-1)/s} = 1$. This factor must be a factor of a, so the verifier will reject in step 7.

Zero-Knowledge. This protocol is obviously zero-knowledge if factoring is easy. In that case the simulator could factor n and then follow the prover's algorithm. Hence we assume that factoring is intractable. In this case the protocol is obviously not perfect zero-knowledge, or even statistical zero-knowledge unless there is some way for the simulator to produce an h of the required form. Since the simulator does not know q, it seems unlikely that it could produce such an h. We make the intractability assumption that finding the factor q of n is random polynomial-time equivalent to distinguishing between random generators and random quasi-generators corresponding to q. This seems reasonable because the known algorithms for testing for primitivity involve factoring $p-1$.

We will be using the definition of weak zero-knowledge given in Definition 8, and the constant c in the following comes from that definition. In place of a quasi-generator, the simulator will produce a random element of Z_p^* which it cannot tell is not a generator (i.e., if r is a factor of a or a small factor of n, where small means less than $\log_2^{c+2} p$, then neither h^r nor $h^{(p-1)/r}$ is the identity). With probability $1 - \log_2^{-c} p$, this element is a generator of Z_p^* (see Appendix B). Thus, under the above intractability assumption, this protocol is computational weak zero-knowledge if finding q is infeasible. Assuming that factoring is hard in general, there exists an infinite subset K of S on which the protocol will be computational weak zero-knowledge. A candidate for a subset of this K is

$$M_\varepsilon = \{n \in S \mid \forall \text{primes } p \mid n \ \exists \text{primes } q_1 \mid p - 1, q_2 \mid p + 1, q_1, q_2 > n^\varepsilon\}$$

since no known factorization algorithm can factor numbers from M_ε in expected polynomial time.

The above discussion gives

Theorem 5. *Assuming the Extended Riemann Hypothesis, then there is a prover-practical, bounded round, interactive proof system for*

$$S = \{n \mid n = q^2 m, q \text{ prime}\},$$

with $CC(N) = O(N^3)$, *where* $N = \lceil \log_2 n \rceil$ *is the size of the input. The prover's auxiliary input tape contains the complete factorization of* n.

Let K *be a subset of* S. *For each* $n \in K$, *we define the distributions* G_n *and* Q_n *as follows. We choose* p *randomly and uniformly such that* $|p| \leq |n|^3$, p *is a prime and* $n \mid p - 1$. *Then choose* g *at random and uniformly from the set of generators of* Z_p^*. *Now look at the two distributions*

$$G_n = \{(g, p)\} \quad \text{and} \quad Q_n = \{(g^q, p)\}.$$

If, for any probabilistic polynomial-time machine D,

$$\forall c, \quad \exists N, \quad \forall n \in K,$$

$$n > N \quad \Rightarrow \quad |\Pr[D(G_n) = 1] - \Pr[D(Q_n) = 1]| \leq \frac{1}{\log^c n},$$

then the protocol is weak zero-knowledge on K.

Notice that the above protocol does not involve any encryption. All previous "natural" zero-knowledge proofs which are neither perfect nor statistical zero-knowledge, such as the zero-knowledge proof in [19] that a graph is 3-colorable, have used some encryption.

4. Open Problems

We would like to find efficient prover-practical zero-knowledge proofs for other problems. In particular, we began working on these problems after David Chaum mentioned the problem of finding an efficient prover-practical zero-knowledge proof that an element g generates a large subgroup modulo a composite number n. That problem is still open. We would also like to eliminate the assumption that $p - 1$ is square-free in the primitivity protocol.

The protocol given here to show that a number is not square-free is zero-knowledge under a reasonable assumption, but not statistical zero-knowledge. A practical statistical or perfect zero-knowledge proof system for this problem would be interesting.

We would also like to find an efficient prover-practical zero-knowledge proof that a number n is square-free.

Acknowledgments

We are very grateful to David Chaum for suggesting the problem mentioned in the last section, to René Peralta for pointing out that proving knowledge of the discrete logarithm is sufficient for step 3 of the primitivity protocol, and to Eric Bach and Kevin McCurley for answering numerous questions on factoring algorithms and the distributions of primes. We would also like to thank Ernie Brickell, Faith Fich, Mark Krentel, Stuart Kurtz, Jeff Shallit, and Janos Simon for helpful discussions. In addition, we would like to thank the anonymous referees for improving the exposition.

Appendix A

In this appendix we give the details of the simulation of the primitivity protocol. This simulation uses ideas from [20]. We describe the situation in which the whole protocol is done in parallel. The last subscript on each variable indicates which of the $\lceil \log_2 p \rceil = n$ repetitions of the sequential protocol it comes from. The simulator excutes the following algorithm when simulating the interaction between the true prover and a fixed verifier V^*.

Run V^* until it has sent $2n + n^2$ numbers $(h_1, h_2, \ldots, h_n, x_1, x_2, \ldots, x_n, h_{1,1}, h_{2,1}, \ldots, h_{n,1}, h_{1,2}, \ldots, h_{n,n})$.
Copy the configuration C of V^* at this point.
Choose randomly and uniformly $(\beta_{1,1}, \beta_{2,1}, \ldots, \beta_{n,1}, \beta_{1,2}, \ldots, \beta_{n,n}) \in \{0, 1\}^n$.
Run V^* from configuration C with input $\beta_{1,1}, \beta_{2,1}, \ldots, \beta_{n,1}, \beta_{1,2}, \ldots, \beta_{n,n}$ until he has sent n^2 numbers $(\hat{r}_{1,1}, \hat{r}_{2,1}, \ldots, \hat{r}_{n,1}, \hat{r}_{1,2}, \ldots, \hat{r}_{n,n})$.
if $\exists i, j$: $h_{i,j} \neq g^{\hat{r}_{i,j}}/h_j^{\beta_{i,j}}$ **then**
  Make a transcript of the communication up to this point and stop.
else (* the simulator has to find the discrete logarithms $(r_1, r_2, \ldots, r_n)$ of $h_1, h_2, \ldots, h_n$.*)
  for $j := 1$ **to** n **do**
    $m := 1$
    while r_j is undefined **do**
      Choose randomly and uniformly $(\gamma_{1,1}, \gamma_{2,1}, \ldots, \gamma_{n,1}, \gamma_{1,2}, \ldots, \gamma_{n,n}) \in \{0, 1\}^{n^2}$
      Run V^* from configuration C with input $\gamma_{1,1}, \gamma_{2,1}, \ldots, \gamma_{n,1}, \gamma_{1,2}, \ldots, \gamma_{n,n}$ until he has sent n^2 numbers $(\tilde{r}_{1,1}, \tilde{r}_{2,1}, \ldots, \tilde{r}_{n,1}, \tilde{r}_{1,2}, \ldots, \tilde{r}_{n,n})$.

if $\exists i$: $h_{i,j} = g^{\tilde{r}_{i,j}}/h_j^{\gamma_{i,j}}$ and $\beta_{i,j} \neq \gamma_{i,j}$ **then** $r_j := (-1)^{\gamma_{i,j}}(\hat{r}_{i,j} - \tilde{r}_{i,j})$
if $m \geq 2^n$ **then** find r_j by brute force from h_j.
$m := m + 1$
Make a transcript of the h_j's, x_j's, $h_{i,j}$'s, $\beta_{i,j}$'s, $\hat{r}_{i,j}$'s, and the r_j's.

Obviously the transcripts produced have the same distribution as would occur with the true prover. The only question is the running time.

Lemma 1. *The above simulator runs in expected polynomial time.*

Proof. It is clear that if the expected number of iterations for the while loop is polynomial, then the running time is expected polynomial time. Note that the brute-force step is only undertaken if the algorithm has already used exponential time. First define

$$p = \Pr\left[\forall i, j\colon h_{i,j} = \frac{g^{\hat{r}_{i,j}}}{h_j^{\beta_{i,j}}}\right].$$

With probability p we get to the while loop. Then we fix $\beta_{1,1}, \beta_{2,1}, \ldots, \beta_{n,n}$, define q_j to be the probability of finding r_j in the only one iteration of the while loop

$$q_j = \Pr\left[\exists i\colon h_{i,j} = \frac{g^{\tilde{r}_{i,j}}}{h_j^{\gamma_{i,j}}} \text{ and } \beta_{i,j} \neq \gamma_{i,j}\right].$$

We observe that p cannot be much bigger than q_j:

$$\begin{aligned}
q_j &\geq \Pr\left[\forall i, j\colon h_{i,j} = \frac{g^{\hat{r}_{i,j}}}{h_j^{\gamma_{i,j}}} \text{ and } \exists i\colon \beta_{i,j} \neq \gamma_{i,j}\right] \\
&= p - \Pr\left[\forall i, j\colon h_{i,j} = \frac{g^{\hat{r}_{i,j}}}{h_j^{\gamma_{i,j}}} \text{ and } \forall i\colon \beta_{i,j} = \gamma_{i,j}\right] \\
&\geq p - \Pr[\forall i\colon \beta_{i,j} = \gamma_{i,j}] \\
&= p - 2^{-n}.
\end{aligned}$$

If X_j is the number of iterations of the while loop in the jth iteration of the for loop, we get

$$\begin{aligned}
E(X_j) &= p\left(\sum_{i=1}^{2^n} i(1 - q_j)^{i-1}q_j + 2^n(1 - q_j)^{2^n}\right) \\
&= p\frac{1 - (1 - q_j)2^n}{q_j} \\
&\leq \frac{p}{q_j} \leq \frac{q_j - 2^{-n}}{q_j} \leq 1 + \frac{2^{-n}}{q_j} \leq 2 \quad \text{if} \quad q_j \geq 2^{-n}.
\end{aligned}$$

If $q_j < 2^{-n}$, then by using that $X_j \leq 2^n$ we get that

$$E(X_j) \leq p2^n \leq (q_j + 2^{-n})2^n \leq 2.$$

Hence the expected number of iterations of the while loop is $\leq 2n$. $\square$

Appendix B

Let C_n be a cyclic group of order n. Let $c \geq 1$ be a constant. Consider the following procedure $\mathscr{A}$:

> Construct the set $S = \{p | p \text{ prime}, p \leq \log^{c+2} n \text{ and } p | n\}$
> **repeat**
> Choose g randomly and uniformly from C_n.
> **until** $\forall p \in S$: $g^{n/p} \neq 1$
> **output** g

Lemma 2. *$\mathscr{A}$ runs in expected polynomial time in $\log^c n$ and $\mathscr{A}$ outputs a nongenerator of C_n with probability $O(1/\log^c n)$.*

Proof. Let $G := \{g \in C_n | g \text{ is a generator}\}$. Now $|G| = \varphi(n) > \Omega(n/(\log \log n))$ [28]. So the expected number of g's picked is $O(\log \log n)$ since every generator passes the test. The construction of S and the test clearly take only polynomial time.

Suppose $n = p_1 p_2 \cdots p_l$, where, for some k, we have that if $i \leq k$, then $p_i \leq \log^{c+2} n$, and if $i > k$, then $p_i > \log^{c+2} n$. Now let $T := \{g | \mathscr{A} \text{ can output } g\} = \{g | g^{n/p_i} \neq 1, i \in [1, k]\}$.

We know that g is a generator if and only if, for all $i \in [1, l]$, $g^{n/p_i} \neq 1$. This shows that

$$T - G \subset \bigcup_{i=k+1}^{l} \{x | x^{n/p_i} = 1\}.$$

Now the cardinality of each term is estimated by

$$|\{x | x^{n/p_i} = 1\}| = \frac{n}{p_i} \leq \frac{n}{\log^{c+2} n}.$$

So we get

$$|T - G| \leq l\left(\frac{n}{\log^{c+2} n}\right) \leq \frac{n}{\log^{c+1} n}.$$

Now the probability that $\mathscr{A}$ outputs a nongenerator is

$$\frac{|T - G|}{|T|} \leq \frac{|T - G|}{|G|}$$

which is $O(1/\log^c n)$. This proves the lemma. $\square$

References

[1] Adleman, L., and M.-D. Huang, Recognizing primes in random polynomial time, *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 462–469.

[2] Adleman, L., K. Manders, and G. Miller, On taking roots in finite fields, *Proc. 18th IEEE Symp. on Foundations of Computer Science*, 1977, pp. 175–178.

[3] Bach, E., How to generate factored random numbers, *SIAM Journal on Computing*, vol. 17, no. 2, April 1988, pp. 179–193.

[4] Bellare, M., S. Micali, and R. Ostrovsky, Perfect zero-knowledge in constant rounds, *Proc. 22nd ACM Symp. on Theory of Computing*, 1990, pp. 482–493.

[5] Benaloh, J., Cryptographic capsules: a disjunctive primitive for interactive protocols, *Advances in Cryptology—Crypto '86 Proceedings*, 1987, pp. 213–222.

[6] Berlekamp, E. Factoring polynomials over large finite fields, *Mathematics of Computations*, vol. 24, 1970, pp. 713–735.

[7] Boppana, R., J. Hastad, and S. Zachos, Does co-NP have short interactive proofs?, *Information Processing Letters*, vol. 25, 1987, pp. 127–132.

[8] Brassard, G., and C. Crépeau, Non-transitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond, *Proc. 27th IEEE Symp. on Foundations of Computer Science*, 1986, pp. 188–195.

[9] Brassard, G., C. Crépeau, and J. M. Robert, All-or-nothing disclosure of secrets, *Advances in Cryptology—Crypto '86 Proceedings*, 1987, pp. 234–238.

[10] Chaum, D., Demonstrating that a public predicate can be satisfied without revealing any information about how, *Advances in Cryptology—Crypto '86 Proceedings*, 1987, pp. 195–199.

[11] Chaum, D., J.-H. Evertse, J. van de Graaf, An improved protocol for demonstrating possession of discrete logarithms and some generalizations, *Advances in Cryptology—Eurocrypt '87 Proceedings*, 1988, pp. 127–141.

[12] Chaum, D., J.-H. Evertse, J. van de Graaf, and R. Peralta, Demonstrating possession of a discrete logarithm without revealing it, *Advances in Cryptology—Crypto '86 Proceedings*, 1987, pp. 200–212.

[13] Davenport, H., *Multiplicative Number Theory*, Markham, Chicago, 1967.

[14] Even, S., A. L. Selman, and Y. Yacobi, The complexity of promise problems with applications to public-key cryptography, *Information and Control*, vol. 61, 1984, pp. 159–173.

[15] Feige, U., A. Fiat, and A. Shamir, Zero-knowledge proofs of identity, *Journal of Cryptology*, vol. 1, no. 2, 1988, pp. 77–94.

[16] Feige, U., and A. Shamir, Zero knowledge proofs of knowledge in two rounds, *Advances in Cryptology—Crypto '89 Proceedings*, 1990, pp. 526–544.

[17] Fortnow, L., The complexity of perfect zero-knowledge, *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 204–209.

[18] Goldreich, O., and E. Kushilevitz, A perfect zero-knowledge proof for a problem equivalent to discrete logarithm, *Advances in Cryptology—Crypto '88 Proceedings*, 1990, pp. 57–70.

[19] Goldreich, O. S. Micali, and A. Wigderson, Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, *Proc. 27th IEEE Symp. on Foundations of Computer Science*, 1986, pp. 174—187.

[20] Goldreich, O., S. Micali, and A. Wigderson, Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, to appear.

[21] Goldwasser, S., and S. Micali, Probabilistic encryption, *Journal of Computer and System Sciences*, vol. 28, 1984, pp. 270–299.

[22] Goldwasser, S., S. Micali, and C. Rackoff, The knowledge complexity of interactive proof systems, *SIAM Journal on Computing*, vol. 18, 1989, pp. 186–208.

[23] Van de Graaf, J., and R. Peralta, A simple and secure way to show the validity of your public key, *Advances in Cryptology—Crypto '87 Proceedings*, 1988, pp. 128–134.

[24] Knuth, D. E., *The Art of Computer Programming*, Vol. 2, Addison-Wesley, Reading, MA, 1969.

[25] Oren, Y., On the cunning power of cheating verifiers: some observations about zero knowledge proofs, *Proc. 28th IEEE Symp. on Foundations of Computer Science*, 1987, pp. 462–471.

[26] Rabin, M. O., Digitalized Signatures and Public-Key Functions as Intractable as Factorization, Technical Report MIT/LCS/TR-212, M.I.T., January 1979.

[27] Rabin, M. O., Probabilistic algorithms in finite fields, *SIAM Journal on Computing*, vol. 9, 1980, pp. 273–280.

[28] Rosser, J. B., and Schoenfeld, L., Approximate formulas for some functions of prime numbers, *Illinois Journal of Mathematics*, vol. 6, 1962, pp. 64–94.

[29] Schwarz, W., Representation of square-free integers, *American Mathematical Monthly*, vol. 73, 1966, pp. 426–427.

[30] Tompa, M., and H. Woll, Random self-reducibility and zero knowledge interactive proofs of possession of information, *Proc. 28th IEEE Symp. on Foundations of Computer Science*, 1987, pp. 472–482.

[31] Wagstaff, S. S., Greatest of the least primes in arithmetic progressions having a given modulus, *Mathematics of Computation*, vol. 33 no. 147, July 1979, pp. 1073–1080.