Practical Zero-Knowledge Proofs: Giving Hints and Using Deficiencies

Joan Boyar, Katalin Friedl and Carsten Lund*
Computer Science Department
University of Chicago

Abstract

New practical 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 super-polynomial in power. A BPP prover with the appropriate trap-door knowledge is sufficient. The proofs are perfect or statistical zero-knowledge in all cases except one.

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 [14], [7], [9]. But 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 non-residuosity [17], were practical; they were efficient and the prover could be BPP if she[1] had the appropriate trap-door knowledge. Other efficient zero-knowledge proofs are given in [8], [10], [11], [13], [18], [25].

In this paper we present a practical zero-knowledge proof for a subproblem of primitivity. This protocol, which shows that an element of the multiplicative group modulo a prime is a generator, only requires that the prover know the complete factorization of $p-1$. Note that the protocol given in [25] is not practical because the prover must be able to 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.

*This research was supported in part by NSA Grant No. MDA904-88-H-2006.

[1]In this paper, it will at times be convenient to think of the verifier as being named Vic, and the prover being named Peggy. This has the advantage that personal pronouns such as "he" and "she" can be used to unambiguously identify one of the parties.

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 155-172, 1990.

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 cryptographic assumption and is thus only computational zero-knowledge rather than perfect or statistical zero-knowledge. The protocol does not, however, involve any bit encryption. 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 non-primitivity, and for membership and non-membership in $\{n|\ n$ and $\varphi(n)$ are relatively prime$\}$.

When we refer to a *practical zero-knoledge proof* we mean one in wich the prover is *BPP* and the proof is direct, i.e. it doesn't involve a transformation to a circuit or an NP-complete problem. Usually such a transformation would involve a very 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.

For a fixed zero-knowledge proof let $CC_k(N)$ be the number of bits communicated to achieve probability of error no more then $(1/2)^k$, where k is the security parameter and N the size of the input. In a typical zero-knowledge proof, one repeats a short protocol k times in order to obtain the required security.

2 The Zero-Knowledge Proofs

2.1 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 $k = \log_2 p$ times:

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 gives a proof of knowledge [13] of r [10].
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 [25], and it still has the problem that the prover needs to be able to take discrete logarithms. We can eliminate this problem, however, by letting the verifier give the prover a hint which will help her to compute the discrete logarithm.

Let us assume that the prover initially has the complete factorization of $p-1$ on her private work tape. Fortunately, 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 will modify the above zero-knowledge proof to include the following steps:

0. The verifier attempts to factor $p-1$ by trial division up to $c\log^c p$. For all nontrivial factors q found, the verifier checks that neither $g^q \pmod p$ nor $g^{(p-1)/q} \pmod p$ is the identity. Here c is an arbitrary constant.

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 gives a proof of knowledge [13] of the discrete logarithm of h [10].

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.

If we assume that $p-1$ is square-free, except possibly for powers of small primes less than $c\log^c p$, the above is a perfect zero-knowledge interactive proof system. Observe that in step 3 the verifier need not prove that x has the correct form because the prover can test this herself from the discrete logarithm. We do, however, have a protocol which proves directly that the form is correct (see Appendix A). It is not immediately obvious, though, that the above protocol is an interactive proof system (i.e. that the verifier should be convinced after the proof is completed), or that the prover can now compute the discrete logarithm (if $p-1$ has many factors).

Let us first show that it is, in fact, an interactive proof system. Suppose that g is not a generator. We will show that in this case the prover will fail to send back the correct r at least 50% of the time. If g is not a generator, then $g = f^k$ for some $f \in Z^*_p$, and $k = tq$ for some prime factor q of $p-1$. By assumption, q^2 does not divide $p-1$. Thus there is another square root r' of x modulo $p-1$ with $r' \equiv -r (\mathrm{mod} q)$, but $r' \equiv r (\mathrm{mod} \frac{p-1}{q})$. This means that there exists an integer s such that $r' = r + s(\frac{p-1}{q})$ and $r' \not\equiv r \pmod{p-1}$. But $g^{r'} = f^{tq(r+s(\frac{p-1}{q}))} = f^{tqr} f^{ts(p-1)} = g^r$. Thus there are at least two distinct square roots of x which are discrete

logarithms of h, so the prover has at best a 50-50 chance of guessing which one the verifier knows.

Now, we will look at the prover's algorithm for finding discrete logarithms given the hint x. The idea is to use the Chinese Remainder Theorem. For each of the prime power factors q of $p-1$, we find the two square roots r_1 and r_2 of x using [2], [6], [21] or [22]. In order to determine which is correct, we compute $(g^{r_1} \cdot h^{-1})^{\frac{p-1}{q}}$ and $(g^{r_2} \cdot h^{-1})^{\frac{p-1}{q}}$. Without loss of generality, suppose $r_1 \equiv r \pmod q$ and $r_2 \equiv -r \pmod q$. Then there exist k_1 and k_2 such that $r_1 = r + k_1 q$ and $r_2 = -r + k_2 q$. Then,

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

and

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

since r was chosen from Z^*_{p-1}. Thus, the prover can simply choose the square root which produces the identity in this formula, and then put all the square roots modulo the different prime power factors together, using the Chinese Remainder Theorem.

The proof that this protocol is in fact a perfect zero-knowledge proof system for primitivity when $p-1$ is "essentially" square-free, follows the lines of [15]. We will sketch some of the ideas for the construction of the simulator. The main idea is to use the verifier (here he can be any BPP-machine) and his proof that he knows r to find this r. In the verifier's proof, the following is done in parallel[2] for $1 \leq i \leq k = \log_2 p$.

1. The verifier randomly and uniformly chooses $r_i \in Z^*_{p-1}$.
2. The verifier computes $h_i \equiv g^{r_i} \pmod p$ and sends it to the prover.
3. The prover chooses $\beta_i \in \{0,1\}$ randomly and sends β_i to the verifier.
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 $\hat{r}_i$
5. The prover checks that $h_i = g^{\hat{r}_i}/h^{\beta_i}$.

This subprotocol may no longer be zero-knowledge when run in parallel, but it does not help a cheating prover. If g is not a generator and $g^m \equiv 1 \pmod p$, then r and $r' = r + lm$ are both discrete logarithms of h. But the verifier could have chosen either r_i or $r_i - lm$. Thus $\hat{r}_i$ will never help the prover distinguish between these possibilities.

[2]This may be done sequentially. We are doing it in parallel to make it clearer that the entire primitivity protocol can be done in parallel.

Now the simulator for the primitivity protocol works as follows. It asks a question $(\beta_1, \beta_2, \ldots, \beta_k)$, and if it does not get a correct answer, 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 also gets a correct answer for this question it can find r, since if $\beta_i' \neq \beta_i$ we have $r = \pm(\hat{r}_i' - \hat{r}_i)$. If it does not get correct answer, then the simulator continues asking questions until it gets a correct answer or it has asked 2^k questions. In the second case it can continue to find r, using brute force. It can be shown that this simulator runs in expected polynomial time for all verifiers. See [15] for more details. Furthermore we can make this a bounded round protocol because this simulator works even if the protocol is run in parallel [15]. Hence we get a bounded round perfect zero-knowledge protocol. If the length of p is N, then the communication cost of the protocol is $O(k^2N)$ because $O(kN)$ bits are communicated in step 3 to achieve error probability not greater then $(1/2)^k$. This gives

Theorem 1 *Let p be a prime and c be an arbitrary constant. Assume that if $q > c\log_2^c p$ is a prime then q^2 does not divide $p-1$. Then there is a practical perfect zero-knowledge, bounded round, interactive proof system for*

$$\{g | < g >= Z_p^*\}.$$

The BPP prover's secret information is the complete factorization of $p-1$. The communication cost of this protocol is $CC_k(N) = O(k^2N)$.

The set of primes for which our proof works is of reasonable size since [24] proved that

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

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 many other cyclic group with known order. Consider, for example, the multiplicative group Z_q^* of the integers modulo $q = p^n$, where p is an odd prime and $n \geq 1$.[3] 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 $g^{p^{n-2}(p-1)} \not\equiv 1 \pmod{p}$. Thus, one can assume that $g \not\equiv h^{tp} \pmod{p}$ for any integer t, and one again only needs to worry about square factors of $p-1$.

[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 $g^{p^{n-2}(p-1)} \not\equiv 1 (\bmod p^n)$ and g is primitive when viewed as an element of the group Z_p^*.

2.2 Are n and $\varphi(n)$ relatively prime?

In the zero-knowledge proof system presented in the previous section, we had to assume that $p-1$ was square-free except for powers of small primes. 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 $p-1=2^k r$, where r is odd, and if r and $\varphi(r)$ are relatively prime, the prover could prove that this is the case and afterwards she could prove primitivity. Unfortunately, it is possible to have r and $\varphi(r)$ 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

$$\{(p,g)|p \text{ is prime}, p-1=2^k r, \text{ where } r \text{ is odd}, \gcd(r,\varphi(r))=1, \text{and} < g >= Z_p^*\}.$$

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 $\log_2 n$ times.

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 n^{th} root of y, thus showing that x has an n^{th} root modulo n.

To see that this works, 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 n^{th} roots also has q^{th} roots. But no more than half of the elements of Z_n^* have q^{th} roots modulo n. If the verifier chooses an x which does not have an n^{th} 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 $\log_2 n$ times exponentially small. When n and $\varphi(n)$ are relatively prime $x \equiv (x^n)^k (\mathrm{mod} n)$ where $k \equiv (n \pmod{\varphi(n)})^{-1} \pmod{\varphi(n)}$. Hence the prover can compute n^{th} roots of x and y.

This is clearly perfect zero-knowledge since the simulator has a 50-50 chance each time of guessing which β the verifier will choose. When the simulator guesses

that $\beta = 0$, he computes y exactly as the prover would, so he has no problem revealing r. When the simulator guesses that $\beta = 1$, he sends $y \equiv r^n \pmod{n}$, so he can give an n^{th} root of y. When he guesses incorrectly, he backs up the transcript tape and tries again. Thus the simulation will be in expected polynomial time. Since y is a random element of Z_n^* whether it is produced by the prover or the simulator, the transcripts produced by the simulator will have the same distribution as those produced by the true prover. Thus this protocol is perfect zero-knowledge. The protocol can furthermore be parallelized following the lines of [4]. The above discussion gives

Theorem 2 *There is a practical perfect zero-knowledge interactive proof system for*

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

with communication cost $CC_k(N) = O(kN)$. The BPP prover's trapdoor information is the number $\varphi(n)$.

Theorem 3 *There is a practical perfect zero-knowledge interactive proof system for*

$$\{(p,g) | p \text{ prime}, p-1 = 2^k r, \text{where } r \text{ is odd}, \gcd(r, \varphi(r)) = 1, \text{and} < g >= Z_p^*\}.$$

The BPP prover's secret information is the complete factorization of $p-1$. The communication cost is $CC_k(N) = O(k^2 N)$.

If n and $\varphi(n)$ are not relatively prime, a prover who knows $\varphi(n)$ can give a practical zero-knowledge proof that they have a common factor, under certain assumptions. One such proof involves repeating the following $\log_2 n$ times. First, the prover sends the verifier a random $x \in Z_n^*$ such that x does not have an n^{th} root. 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 n^{th}-nonresidue x, and it's 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's unlikely that x would be a q^{th}-residue. In this case, the protocol would be statistical zero-knowledge.

2.3 Nongenerators

Suppose p is a prime and g is not a generator of Z_p^*. If the prover knows a $t < p-1$ such that $g^t \pmod{p} \equiv 1$, then she can give a practical statistical zero-knowledge proof that g is not a generator. The proof is statistical zero-knowledge if $s = \frac{p-1}{t}$ is large. The major advantage of the protocol given here over that in [25] 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 work tape; and the prover is attempting to convince the verifier that g is not a generator modulo p. Our proof is based on the fact that for every integer r, $g^r \equiv g^{r+tl} \pmod{p}$ if l is an integer, 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 $\log_2 p$ independent repetitions of the following:

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 $[0, \lceil \frac{s-3}{2} \rceil]$. If $\beta = 1$, the prover chooses a random z uniformly from $[\lfloor \frac{s+1}{2} \rfloor, s-1]$.
5. The prover sends the verifier $r' = r + zt$ who checks that $h \equiv g^{r'} \pmod{p}$ and that $r' \in [1, \frac{p-1}{2}]$ if $\beta = 0$, or that $r' \in [\frac{p-1}{2} + 1, p-1]$ otherwise.

Notice that in step 5 the prover is revealing a discrete logarithm of h which is less than $\frac{p-1}{2}$ if the verifier's challenge was $\beta = 0$, or greater than $\frac{p-1}{2}$ if $\beta = 1$. If g were 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 the correct response. The communication cost of this protocol is $CC_k(N) = O(kN)$.

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 $g^r \pmod{p}$ 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 cannot answer, he will backtrack the verifier to the point of choosing r and try another one. so the simulation is expected polynomial time. Both the prover and the simulator choose h to be a random element of the subgroup generated by g, but 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 $[\frac{p-1}{2} - \frac{t}{2} + 1, \frac{p-1}{2} + \frac{t}{2}]$ 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 $\{1,\ldots,p-1\}$, $|\sum_{x\in X} P(x) - \sum_{x\in X} S(x)| \leq \frac{1}{s}$. Hence for the whole protocol the distributions differ by at most $\frac{\log_2 p}{s}$. Thus this protocol is statistical zero-knowledge on subsets of S of the form

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

where f is superpolynomial.

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 is large. 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 large and small, we can't 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 BPP-distinguisher with probability greater than $1 - \frac{1}{n^c}$ for every c for n sufficiently large. In our definition, we give c to the simulator, which then runs in time polynomial in n^c. Hence the simulator is BPP for fixed c. Otherwise this definition is identical to Oren's [20], and we are using similar notation.

Definition 1 *Let (P,V) be a interactive proof system for L. Then (P,V) is weak zero-knowledge if*

$$\forall V^* : \exists M_{V^*} : \forall x \in L : \forall y : \forall D \in BPP : \forall c :$$
$$|\Pr[D((< P(x), V^*(x,y) >) = 0] - \Pr[D(M_{V^*}(c,x,y)) = 0]| \leq \frac{1}{|x|^c}.$$

It is weak statistical zero-knowledge if, for any subset T of transcripts:

$$\forall V^* : \exists M_{V^*} : \forall x \in L : \forall y : \forall c :$$
$$|\Pr[< P(x), V^*(x,y) >\in T] - \Pr[M_{V^*}(c,x,y)) \in T]| \leq \frac{1}{|x|^c}.$$

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 $s \geq \log^{c+1} n$. If it finds s and hence t, it proceeds as the real prover would; otherwise it proceeds as the old simulator would.

With this new definition of zero-knowledge we can also remove the assumption, in the protocol in [25] for the same problem, that one generator is publicly known. We can let the prover give the verifier a random generator. This is practical weak zero-knowledge because the simulator can find a generator with probability $1 - \log^{-c} n$ in time polynomial in $\log^c n$. See Appendix B.

The above discussion gives

Theorem 4 *There is a practical interactive proof system for*

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

and it is statistical zero-knowledge on

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

where f is superpolynomial. This protocol has communication cost $CC_k(N) = O(kN)$. The BPP prover's secret knowledge is t.

Using our new definition we get:

Theorem 5 *There is a 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 $CC_k(N) = O(kN)$. The BPP prover's secret information is t.

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 a power of an odd prime, 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].

In the parallel protocol, Peggy will first choose randomly and uniformly $x \in Z^*_{p-1}$, compute $f \equiv g^x \pmod{p}$, and send f to Vic. Next, Vic chooses all his challenges $(\beta_1, \beta_2, \ldots, \beta_{\log n})$ 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 $(t_1, t_2, \ldots, t_{\log n})$ to Peggy, who now does steps 1 and 2 from the original protocol and sends $(h_1, h_2, \ldots, h_{\log n})$ to Vic. Vic now reveals his challenges by sending $(s_1, s_2, \ldots, s_{\log n})$. Finally Peggy, after checking that Vic did not cheat will send x and $(r'_1, r'_2, \ldots, r'_{\log n})$. Vic checks that $f = g^x$, checks that $x \in Z^*_{p-1}$ and performs the checks corresponding to step 5.

To see that this is still a proof system, observe that the two different commitments come from the same distribution since $x \in Z^*_{p-1}$. Thus receiving these commitments earlier is no help to Peggy.

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 had just made his commitments. Now the simulator forms its h_i's so that it can answer Vic's questions. If Vic reveals the same old questions, the simulator can answer them. If he reveals another set of questions, the simulator know s, s' such that $g^s = f^{s'}$. This gives

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

It is easy to see that, since Peggy chose x randomly, $s - xs' \pmod{p-1}$ is a random multiple of t, the order of g. If we run the above procedure twice, either the simulator will be able to successfully perform the simulation or it will get two independent, random multiples of t, at and $a't$. We know from [19] that $\Pr[\gcd(at, a't) = t] = 6/\pi^2$. Thus the simulator will succed in expected polynomial time.

If the modulus has more than one prime factor or is a large power of two, no elements would be generators. One 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, one could give a zero-knowledge proof that g only generates a small subgroup.

2.4 Does n have a square factor?

Recall that the protocol given above for proving that an element is a generator of the multiplicative group modulo a prime p only works when $p-1$ is "essentially" square-free. The only problems that arise when $p-1$ has large square factors is that some nongenerators may look like generators. In fact, if g is a generator, the prover can make a nongenerator $g' = g^q$ look like a generator if $p - 1 = 2^l p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ and $q = p_1^{f_1} p_2^{f_2} \cdots p_k^{f_k}$ where $0 \leq f_i < e_i$ for all i (assuming that $p-1$ and q are such that the verifier cannot determine himself that g' is not a generator). The prover can do this because the prover's algorithm for finding the discrete logarithms given the hint x will still work. Let us call any g' of this form a quasi-generator. Note that if $f_i = e_i$ for any i, then there will be two distinct square roots of x which will work as discrete logarithms, so the protocol is in fact a perfect zero-knowledge proof that g is a quasi-generator, regardless of whether or not $p-1$ is square-free. In the case that $p-1$ is square-free, all quasi-generators are actually generators.

It is possible to use this deficiency in the proof system for primitivity to show that an integer n is not square-free. The set we are concerned with is

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

The integer n is available to both the prover and the verifier; the complete factorization of n is initially on the prover's private work tape; and the prover is attempting to convince the verifier that n has a nontrivial square factor. To do this, the prover first finds a prime $p = an + 1$. Assuming the Extended Riemann Hypothesis, one can try random a's which are less than n^2 and expect to find such a prime in time $O(\log n)$.

To see this, consider the following from [12](pp.129, 136). Assuming the Extended Riemann Hypothesis,

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

where

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

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

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

We have from [23] that $\varphi(n) \geq C(n/\log\log n)$; hence if $x = n^3$ the above is greater than

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

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

To find p, one can use Bach's method [3] to produce an appropriate a randomly, along with the complete factorization of a. Since this protocol would be unnecessary if the verifier could find q, we can assume that q does not divide a.

Another way to find an appropriate p is by trying $n+1, 2n+1, 3n+1, \ldots$ until we find a prime. Wagstaff [26] has given an 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.

After finding such a prime, the prover will produce an element $h \in Z_p^*$ which is of the form $g^q \pmod p$ for some generator g. The prover can then use the above protocols to show, first, that h is not a generator, and, second, that h is a quasi-generator. Of course, since h is not a generator and it is a quasi-generator, $p-1$ is not square-free. But the verifier can check that $h^{(p-1)/r} \not\equiv 1 \pmod p$ for any prime factor r of a. Thus, the square factor in question must be a factor of n, proving that n does, in fact, have a square factor.

This 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 will make the cryptographic 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 $p-1$. In place of the quasi-generators 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 $\log_2^{c+1} p$, then neither h^r nor $h^{(p-1)/r}$ is the identity). With probability $\log_2^{-c} p$ this element is a generator of Z_p^* (see Appendix B). Thus, under the above cryptographic assumption, this protocol is computational zero-knowledge if the verifier can't find q. Now under the usual

assumption, that factoring is hard in general, there exists a infinite subset K of S on, which the protocol will be computational weak zero-knowledge. A candidate for a subset of this K is

$$M_\epsilon = \{n |\ \forall \text{primes } p | n\ \exists \text{primes } q_1 | p-1, q_2 | p+1, q_1, q_2 > n^\epsilon\}$$

since no known factorization algorithm can factor numbers from M_ϵ in expected polynomial time. The protocol does not, however, involve any bit encryption. All previous "natural" zero-knowledge proofs which are neither perfect nor statistical zero-knowledge have used bit encryptions.

The above discussion gives

Theorem 6 *Assuming the Extended Riemann Hypothesis there is a practical interactive proof system for*

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

with $CC_k(N) = O(k^2 N)$. The BPP prover secret information is the complete factorization of n.

Let K be a subset of S. For each $n \in K$, we will define the distributions G_n and Q_n as follows. We will 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

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

If

$$\forall D \in BBP\ \forall c\ \exists N\ \forall n \in K : n > N \Rightarrow |\Pr[D(G_n) = 1] - \Pr[D(Q_n) = 1]| \leq \frac{1}{\log^c n}$$

then the protocol is weak zero-knowledge on K.

3 Open Problems

One would like to find practical zero-knowledge proofs for other problems. In particular, we began working on these problems after David Chaum mentioned the problem of finding a 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 "essentially" square-free in the primitivity protocol.

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

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

4 Acknowledgements

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.

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, personal communication.

[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] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] Davenport, H., *Multiplicative Number Theory*, Markham Publishing Company, 1967.

[13] Feige, U., A. Fiat, and A. Shamir, *Zero-knowledge proofs of identity*, Journal of Cryptology, 1(2), 1988, pp. 77-94.

[14] 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.

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

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

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

[18] 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.

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

[20] 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.

[21] Rabin, M.O., *Digitalized signatures and public-key functions as intractable as factorization*, Technical Report MIT/LCS/TR-212, M.I.T., January 1979.

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

[23] Rosser, J. B., and Schoenfeld, L., *Approximate Formulas for some Functions of Prime Numbers*, Illinois Journal of Math. vol. 6, 1962, pp. 64-94.

[24] Schwarz, W., in American Math. Monthly, vol. 73, 1966, pp. 426-427.

[25] 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.

[26] 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.

Appendix A

In this appendix we show how the verifier in the primitivity protocol can prove that for $h \in Z_p^*$ and $x \in Z_{p-1}^*$ he "knows" [13] an r such that $g^r \equiv h \pmod{p}$ and $r^2 \equiv x \pmod{p-1}$. Both the verifier and prover can compute $a \equiv g^x \pmod{p} \equiv h^r \pmod{p}$. Hence if the verifier can show that h and a were formed by raising g and h, respectively, to the same power, it will have shown the correct form of h and x. Rather than simply presenting a zero-knowledge proof of knowledge for this problem, we will present a practical zero-knowledge proof of knowledge for something more general. Suppose a prover and verifier (the original verifier will temporarily be acting as a prover) are given the following elements of a commutative group u, v, $a_1, a_2, \ldots, a_n$, and $b_1, b_2, \ldots, b_n$, and that the prover wants to show that she knows integers $e_1, e_2, \ldots, e_n$ such that

$$u = a_1^{e_1} a_2^{e_2} \ldots a_n^{e_n}$$

and

$$v = b_1^{e_1} b_2^{e_2} \ldots b_n^{e_n}.$$

In this zero-knowledge proof, the following will be repeated a number of times equal to the length of the input.

1. The prover chooses random $c_1, c_2, \ldots, c_n$ in the range $[1 \ldots t]$, where t is the order of the group.
2. The prover computes $y = a_1^{c_1} a_2^{c_2} \ldots a_n^{c_n}$ and $z = b_1^{c_1} b_2^{c_2} \ldots b_n^{c_n}$.
3. The prover computes $u' = uy$ and $v' = vz$ and sends these values to the verifier.
4. The verifier chooses $\beta \in \{0, 1\}$ randomly with equal probabilities and sends β to the prover.
5. If $\beta = 0$, the prover reveals $c_1, c_2, \ldots, c_n$, so the verifier can check that u' and v' were formed correctly. If $\beta = 1$, the prover reveals $e_1 + c_1, e_2 + c_2, \ldots, e_n + c_n$, so the verifier can check that the prover knows the same information about u' and v' that she was supposed to know about u and v.

This is clearly a proof of knowledge because an observer seeing the prover respond to both challenges for the same u' and v' could compute $e_1, e_2, \ldots, e_n$ by subtracting. It is also zero-knowledge if the order of the group is known as it would be for the group Z_p^* with which we are mainly concerned. To see this, consider the following simulator which produces transcripts of the proof with exactly the same distribution as would be produced with the legitimate prover. First, the simulator flips a coin. If the result of this coin flip is "heads", the simulator is guessing that the verifier will send $\beta = 0$. Thus, the simulator goes through exactly those steps

the prover would in forming u' and v'. Since we are assuming that the simulator knows the order of the group, this is easy. If, when the simulator runs the program for the verifier with this u' and v', the verifier sends $\beta = 0$, the simulator has no problem in revealing the required values. If, however, the verifier sends $\beta = 1$, the simulator will back up the tape over these current u' and v' and will try again with another coin flip. If the result of the coin flip is "tails", the simulator is guessing that the verifier will send $\beta = 1$, so it will produce the random c_i's, the y and the z, but it will send the verifier y and z for u' and v'. When it receives $\beta = 1$ from the verifier, it can simply reveal the c_i's. Of course, if it receives $\beta = 0$, it will have to back up the tape and try again. But it has a 50-50 chance of succeeding each time so the simulation is expected polynomial time, and it is easy to check that it produces transcripts with exactly the same distribution as those produced with the true prover. Thus, this proof system is perfect zero-knowledge.

Appendix B

Let C_n be a cyclic group of order n.

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

Theorem 7 *Prob(g is not a generator)* $< \frac{1}{\log^c n}$

proof :
Suppose $n = p_1 p_2 \dots p_k q_1 q_2 \dots q_l$ where each $p_i \leq \log^{c+1} n$ and each $q_i > \log^{c+1} n$.
Let $n' = p_1 p_2 \dots p_k$ and $T = \{g|$ the procedure can output $g\} = \{g| \ \forall i : g^{n/p_i} \neq 1\}$.
Now for $d|n$ define $A_d = \{x|$ order$(x) = d\}$. $\cup_{d|n} A_d = C_n$ and $|A_d| = \varphi(d)$,

$$T = \bigcup_{\substack{d|n \\ n'|d}} A_d.$$

So we get that

$$|T| = \sum_{d|\frac{n}{n'}} \varphi(n'd) = \varphi(n') \sum_{d|\frac{n}{n'}} \varphi(d).$$

Furthermore we have that

$$|G| = |\{g| < g >= C_n\}| = \varphi(n) = \varphi(n')\varphi(n/n').$$

Hence we can assume that $k = 0$, i.e. that $q|n$ implies that $q > \log^{c+1} n$.

$$T \setminus G = \bigcup_{i=1}^{l} \{x | x^{n/q_i} = 1\},$$

where the cardinality of each term is estimated by

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

So we get

$$|T \setminus G| \leq l(n/\log^{c+1} n) \leq n/\log^{c} n.$$

This proves the theorem. □