CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 20
November 13, 2017

Formalizing Zero Knowledge

Computational Knowledge

What does Bob learn from Alice?

We have seen several examples of zero knowledge proofs but no careful definition of what it means to be “zero knowledge”.

The intuition that “Bob learns nothing from Alice” surely isn’t true.

After running the FFS protocol, for example, Bob learns the quadratic residue x that Alice computed in the first step.

He didn’t know x before, nor did he and Alice know any quadratic residues in common other than the public number v.

By zero knowledge, we want to capture the notion that Bob learns nothing that might be useful in turning an intractable computation into a tractable one.

A general client process for interacting with Alice

Consider an arbitrary feasible algorithm for performing some computation, i.e., suppose Mallory is trying to compute some random function $f(z)$.

We regard Mallory as a networked probabilistic polynomial time (NPPT) Turing machine. Mallory has an input tape, an output tape, and a network interface that allows her to talk to Alice.

From time to time, Mallory plays Bob's role in some zero-knowledge protocol with Alice, say FFS for definiteness.

If Mallory successfully computes $f(z)$ in this way, we say that she does so *with Alice's help*.

A view of Mallory

In more detail, here's how Mallory's computation of $f(z)$ works:

- z is placed on Mallory's input tape.
- From time to time, Mallory contacts Alice and begins FFS.
- Alice sends a number x that Mallory reads.
- Later, Mallory sends a bit b to Alice.
- Later still, Mallory reads Alice's response y.
- Mallory continues computing, possibly running FFS again.
- If Mallory halts, the answer $f(z)$ is found on her output tape.

A Mallory-simulator

A Mallory-simulator, whom we'll call Sam, is a program like Mallory, but he lacks network access and can't talk to Alice.

Alice's protocol is *zero knowledge* if every random function $f(z)$ that can be computed by some Mallory with Alice's help can also be computed by some Sam without Alice's help.

In other words, whatever Mallory does with the help of Alice, Sam can do alone.

Why Alice's secret is safe

We assume that finding square roots modulo $n = pq$ is hard. More precisely, we assume that no PPT algorithm can find any square root $x \in \sqrt{y} \pmod{n}$ of a quadratic residue y with non-negligible success probability.

Theorem

Under the above assumption, no NPPT algorithm Mallory (n, v) that interacts with Alice using FFS can find her secret s with non-negligible success probability.

It follows that Alice's secret is safe when she runs FFS.

Proof that Alice's secret remains private

Suppose some NPPT Mallory with inputs n and v is able to output s with non-negligible success probability with Alice's help.

Because FFS is zero-knowledge, one can construct a Mallory simulator Sam that outputs s with the same success probability, without Alice's help.

But this contradicts the assumption that taking square roots is hard, since if Sam were to exist, he could be turned into a general-purpose square-root algorithm.

We conclude that Alice's secret remains secure when she runs FFS.

$\square$

Constructing a simulator

To show that a particular interactive protocol is zero knowledge, one must show how to construct Sam for arbitrary NPPT programs Mallory.

Here's a sketch of how to generate a triple (x, b, y) for the FFS protocol.

$b = 0$: Sam generates x and y the same way Alice does—by taking $x = r^2 \bmod n$ and $y = r \bmod n$.

$b = 1$: Sam chooses y at random and computes $x = y^2 v \bmod n$.

What he can't do (without knowing Alice's secret) is to generate both triples for the same value x.

A simulator for FFS

Here's the code for Sam:

1. Simulate Mallory until she requests a value from Alice.
2. Save Mallory's state as Q.
3. Choose a random value $\hat{b} \in \{0, 1\}$.
4. Generate a valid random triple $(x, \hat{b}, y)$.
5. Pretend that Alice sent x to Mallory.
6. Continue simulating Mallory until she is about to send a value b to Alice.
7. If $b \neq \hat{b}$, reset Mallory to state Q and return to step 3.
8. Otherwise, continue simulating Mallory until she requests another value from Alice. Pretend that Alice sent her y and continue.
9. Continue simulating Mallory in this way until she halts.

Properties of the simulator

The probability that $b = \hat{b}$ in step 7 is $1/2$; hence, the expected number of times Sam executes lines 3–7 is only 2.

Sam outputs the same answers as Mallory with the same probability distribution. Requires some work to show.

Hence, the FFS protocol is zero knowledge.

Note that this proof depends on Sam's ability to generate triples of both kinds without knowing Alice's secret.

Composing Zero-Knowledge Proofs

Serial composition

One round of the simplified FFS protocol has probability 0.5 of error. That is, an Alice-impostor can fool Bob half the time.

This is unacceptably high for most applications.

Repeating the protocol t times reduces error probability to $1/2^t$.

Taking $t = 20$, for example, reduces the probability of error to less than on in a million.

The downside of such *serial repetition* is that it also requires t round trip messages between Alice and Bob (plus a final message from Alice to Bob).

Parallel composition of zero-knowledge proofs

One could run t executions of the protocol in parallel.

Let (x_i, b_i, y_i) be the messages exchanged during the i^{th} execution of the simplified FFS protocol, $1 \leq i \leq t$.

In a *parallel execution*,

- Alice sends $(x_1, \ldots, x_t)$ to Bob,
- Bob sends $(b_1, \ldots, b_t)$ to Alice,
- Alice sends $(y_1, \ldots, y_t)$ to Bob,
- Bob checks the t sets of values he has received and accepts only if all checks pass.

Simulation proof does not extend to parallel execution

A parallel execution is certainly attractive in practice, for it reduces the number of round-trip messages to only $1\frac{1}{2}$.

The downside is that the resulting protocol may not be zero knowledge by our definition.

Intuitively, the important difference is that Bob gets to know all of the x_i's before choosing the b_i's.

Problem extending the simulator to the parallel case

While it seems implausible that this would actually help a cheating Bob to compromise Alice's secret, the simulation proof used to show that a protocol is zero knowledge no longer works.

To extend the simulator construction to the parallel composition:

- First Sam would have to guess $(\hat{b}_1, \ldots \hat{b}_t)$.
- He would construct the x_i's and y_i's as before.
- When Mallory's program reaches the point that she generates the b_i's, the chance is very high that some of Sam's guesses were wrong, and he will be forced to go back and try again.

  Indeed, the probability that all t initial guesses are correct is only $1/2^t$.

Quadratic Residues Revisited

QR reprise

Quadratic residues play a key role in the Feige-Fiat-Shamir zero knowledge authentication protocol.

They can also be used to produce a secure probabilistic cryptosystem and a cryptographically strong pseudorandom bit generator.

Before we can proceed to these protocols, we need some more number-theoretic properties of quadratic residues.

Euler criterion

The Euler criterion gives a feasible method for testing membership in QR_p when p is an odd prime.

Theorem (Euler Criterion)

An integer a is a non-trivial[1] *quadratic residue modulo an odd prime p iff*

$$a^{(p-1)/2} \equiv 1 \pmod{p}.$$

[1] A non-trivial quadratic residue is one that is not equivalent to 0 (mod p).

Proof of Euler Criterion

Proof in forward direction.
Let $a \equiv b^2 \pmod{p}$ for some $b \not\equiv 0 \pmod{p}$. Then

$$a^{(p-1)/2} \equiv (b^2)^{(p-1)/2} \equiv b^{p-1} \equiv 1 \pmod{p}$$

by Euler's theorem, as desired. □

Proof of Euler Criterion (continued)

Proof in reverse direction.

Suppose $a^{(p-1)/2} \equiv 1 \pmod{p}$. Clearly $a \not\equiv 0 \pmod{p}$. We find a square root b of a modulo p.

Let g be a primitive root of p. Choose k so that $a \equiv g^k \pmod{p}$, and let $\ell = (p-1)k/2$. Then

$$g^\ell \equiv g^{(p-1)k/2} \equiv (g^k)^{(p-1)/2} \equiv a^{(p-1)/2} \equiv 1 \pmod{p}.$$

Since g is a primitive root and $g^\ell \equiv 1 \pmod{p}$, then $\phi(p) \mid \ell$. Hence, $\ell/\phi(p) = \ell/(p-1) = k/2$ is an integer.

Let $b = g^{k/2}$. Then $b^2 \equiv g^k \equiv a \pmod{p}$, so b is a non-trivial square root of a modulo p, as desired. □

A hard problem associated with quadratic residues

Let $n = pq$, where p and q are distinct odd primes.

Recall that each $a \in \mathrm{QR}_n$ has 4 square roots, and $1/4$ of the elements in $\mathbf{Z}_n^*$ are quadratic residues.

Some elements of $\mathbf{Z}_n^*$ are easily recognized as non-residues, but there is a subset of non-residues (which we denote by Q_n^{00}) that are *hard to distinguish* from quadratic residues without knowing p and q.

Quadratic residues modulo $n = pq$

Let $n = pq$, p, q distinct odd primes.

We divide the numbers in $\mathbf{Z}_n^*$ into four classes depending on their membership in QR_p and QR_q.[2]

- Let $Q_n^{11} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QR}_p \cap \mathrm{QR}_q\}$.
- Let $Q_n^{10} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QR}_p \cap \mathrm{QNR}_q\}$.
- Let $Q_n^{01} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QNR}_p \cap \mathrm{QR}_q\}$.
- Let $Q_n^{00} = \{a \in \mathbf{Z}_n^* \mid a \in \mathrm{QNR}_p \cap \mathrm{QNR}_q\}$.

Under these definitions, $$\mathrm{QR}_n = Q_n^{11}$$

$$\mathrm{QNR}_n = Q_n^{00} \cup Q_n^{01} \cup Q_n^{10}$$

[2]To be strictly formal, we classify $a \in \mathbf{Z}_n^*$ according to whether or not $(a \bmod p) \in \mathrm{QR}_p$ and whether or not $(a \bmod q) \in \mathrm{QR}_q$.

Quadratic residuosity problem

Definition (Quadratic residuosity problem)

The *quadratic residuosity problem* is to decide, given $a \in Q_n^{00} \cup Q_n^{11}$, whether or not $a \in Q_n^{11}$.

Fact

There is no known feasible algorithm for solving the quadratic residuosity problem that gives the correct answer significantly more than 1/2 the time for uniformly distributed random $a \in Q_n^{00} \cup Q_n^{11}$, unless the factorization of n is known.

Securely encrypting single bits

Goldwasser and Micali devised a probabilistic public key cryptosystem based on the assumed hardness of the quadratic residuosity problem that allows one to securely encrypt single bits.

The idea is to encrypt a "0" by a random residue of QR_n and a "1" by a random non-residue in Q_n^{00}. Any ability to decrypt the bit is tantamount to solving the quadratic residuosity problem.

Goldwasser-Micali probabilistic cryptosystem

Key Generation

The public key consists of a pair $e = (n, y)$, where $n = pq$ for distinct odd primes p, q, and y is any member of Q_n^{00}.

The private key consists of the triple $d = (n, y, p)$.

The message space is $\mathcal{M} = \{0, 1\}$. (Single bits!)

The ciphertext space is $\mathcal{C} = Q_n^{00} \cup Q_n^{11}$.

Goldwasser-Micali probabilistic cryptosystem (cont.)

Encryption

To encrypt $m \in \mathcal{M}$, Alice chooses a random $r \in \mathbf{Z}_n^*$ and sets $a = r^2 \bmod n$. The result a is a random element of $\mathrm{QR}_n = Q_n^{11}$.

If $m = 0$, set $c = a$ (which is in Q_n^{11}).
If $m = 1$, set $c = ay \bmod n$ (which is in Q_n^{00}).

Decryption

Bob, knowing the private key p, can use the Euler Criterion to quickly determine whether or not $c \in \mathrm{QR}_p$ and hence whether $c \in Q_n^{11}$ or $c \in Q_n^{00}$, thereby determining m.

Goldwasser-Micali probabilistic cryptosystem (cont.)

Security
Eve's problem of finding m given c is equivalent to the problem of testing if $c \in Q_n^{11}$, given that $c \in Q_n^{00} \cup Q_n^{11}$.

This is just the quadratic residuosity problem, assuming the ciphertexts are uniformly distributed. One can show:

- Every element of Q_n^{11} is equally likely to be chosen as the ciphertext c in case $m = 0$;
- Every element of Q_n^{00} is equally likely to be chosen as the ciphertext c in case $m = 1$.

If the messages are also uniformly distributed, then any element of $Q_n^{00} \cup Q_n^{11}$ is equally likely to be the ciphertext.

Important facts about quadratic residues

1. If p is odd prime, then $|\mathrm{QR}_p| = |\mathbf{Z}_p^*|/2$, and for each $a \in \mathrm{QR}_p$, $|\sqrt{a}| = 2$.
2. If $n = pq$, $p \neq q$ odd primes, then $|\mathrm{QR}_n| = |\mathbf{Z}_n^*|/4$, and for each $a \in \mathrm{QR}_n$, $|\sqrt{a}| = 4$.
3. Euler criterion: $a \in \mathrm{QR}_p$ iff $a^{(p-1)/2} \equiv 1 \pmod{p}$, p odd prime.
4. If p is odd prime, $a \in \mathrm{QR}_p$, can feasibly find $y \in \sqrt{a}$. (See appendix.)
5. If $n = pq$, $p \neq q$ odd primes, then distinguishing Q_n^{00} from Q_n^{11} is believed to be infeasible. Hence, infeasible to find $y \in \sqrt{a}$. Why?
   If not, one could attempt to find $y \in \sqrt{a}$, check that $y^2 \equiv a \pmod{n}$, and conclude that $a \in Q^{11}$ if successful.

Appendix: Finding Square Roots

Finding square roots modulo prime $p \equiv 3 \pmod 4$

The Euler criterion lets us test membership in QR_p for prime p, but it doesn't tell us how to quickly find square roots. They are easily found in the special case when $p \equiv 3 \pmod 4$.

Theorem

Let $p \equiv 3 \pmod 4$, $a \in \mathrm{QR}_p$. Then $b = a^{(p+1)/4} \in \sqrt{a} \pmod p$.

Proof.

$p + 1$ is divisible by 4, so $(p+1)/4$ is an integer. Then

$$b^2 \equiv (a^{(p+1)/4})^2 \equiv a^{(p+1)/2} \equiv a^{1+(p-1)/2} \equiv a \cdot 1 \equiv a \pmod p$$

by the Euler Criterion. □

Finding square roots for general primes

We now present an algorithm due to D. Shanks[3] that finds square roots of quadratic residues modulo any odd prime p.

[3]Shanks's algorithm appeared in his paper, "Five number-theoretic algorithms", in Proceedings of the Second Manitoba Conference on Numerical Mathematics, Congressus Numerantium, No. VII, 1973, 51–70. Our treatment is taken from the paper by Jan-Christoph Schlage-Puchta", "On Shank's Algorithm for Modular Square Roots", *Applied Mathematics E-Notes, 5* (2005), 84–88.

Shank's algorithm

Let p be an odd prime. Write $\phi(p) = p - 1 = 2^s t$, where t is odd.
(Recall: s is # trailing 0's in the binary expansion of $p - 1$.)

Because p is odd, $p - 1$ is even, so $s \geq 1$.

A special case

In the special case when $s = 1$, then $p - 1 = 2t$, so $p = 2t + 1$.

Writing the odd number t as $2\ell + 1$ for some integer ℓ, we have

$$p = 2(2\ell + 1) + 1 = 4\ell + 3,$$

so $p \equiv 3 \pmod{4}$.

This is exactly the case that we handled above.

Overall structure of Shank's algorithm

Let $p - 1 = 2^s t$ be as above, where p is an odd prime.

Assume $a \in \mathrm{QR}_p$ is a quadratic residue and $u \in \mathrm{QNR}_p$ is a quadratic non-residue.

We can easily find u by choosing random elements of $\mathbf{Z}_p^*$ and applying the Euler Criterion.

The goal is to find x such that $x^2 \equiv a \pmod{p}$.

Shanks's algorithm

```
1.  Let $s$, $t$ satisfy $p - 1 = 2^s t$ and $t$ odd.
2.  Let $u \in \mathrm{QNR}_p$.
3.  $k = s$
4.  $z = u^t \bmod p$
5.  $x = a^{(t+1)/2} \bmod p$
6.  $b = a^t \bmod p$
7.  while $(b \not\equiv 1 \pmod{p})$ {
8.      let $m$ be the least integer with $b^{2^m} \equiv 1 \pmod{p}$
9.      $y = z^{2^{k-m-1}} \bmod p$
10.     $z = y^2 \bmod p$
11.     $b = bz \bmod p$
12.     $x = xy \bmod p$
13.     $k = m$
14. }
15. return $x$
```

Loop invariant

The congruence

$$x^2 \equiv ab \pmod{p}$$

is easily shown to be a loop invariant.

It's clearly true initially since $x^2 \equiv a^{t+1}$ and $b \equiv a^t \pmod{p}$.

Each time through the loop, a is unchanged, b gets multiplied by y^2 (lines 10 and 11), and x gets multiplied by y (line 12); hence the invariant remains true regardless of the value of y.

If the program terminates, we have $b \equiv 1 \pmod{p}$, so $x^2 \equiv a$, and x is a square root of $a \pmod{p}$.

Termination proof (sketch)

The algorithm terminates after at most $s - 1$ iterations of the loop.

To see why, we look at the orders[4] of b and $z \pmod{p}$ and show the following loop invariant:

> *At the start of each loop iteration (before line 8),* $\mathrm{ord}(b)$ *is a power of 2 and* $\mathrm{ord}(b) < \mathrm{ord}(z) = 2^k$*.*

After line 8, $m < k$ since $2^m = \mathrm{ord}(b) < 2^k$. Line 13 sets $k = m$ for the next iteration, so k decreases on each iteration.

The loop terminates when $b \equiv 1 \pmod{p}$. Then $\mathrm{ord}(b) = 1 < 2^k$, so $k \geq 1$. Hence, the loop is executed at most $s - 1$ times.

[4] Recall that the order of an element g modulo p is the least positive integer k such that $g^k \equiv 1 \pmod{p}$.