CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 18
November 4, 2013

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

We regard Mallory as a probabilistic Turing machine with input tape and output tape.

- z is placed on the input tape at the beginning.
- If Mallory halts, the contents of the output tape is the answer.
- Mallory can also play Bob's role in some zero-knowledge protocol, say FFS for definiteness.
- During the computation, Mallory can read the number x that Alice sends at the start of FFS.
- Later, he can send a bit b to Alice.
- Later still, he can read the response y from Alice.
- After that, he computes and produces the answer, $f(z)$.

A Mallory-simulator

A Mallory-simulator, whom we'll call Sam, is a program like Mallory except he is not on the internet and can't talk to Alice.

Alice's protocol is *zero knowledge* if for every Mallory, there is a Mallory-simulator Sam that computes the same random function $f(z)$ as Mallory.

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

The logical connection with knowledge

If Mallory computes some function with Alice's help (such as writing a square root of v to the output tape), then Sam can also do that without Alice's help.

Under the assumption that taking square roots is hard, Sam couldn't do that; hence Mallory also couldn't do that, even after talking with Alice.

We conclude that Alice doesn't release information that would help Mallory to compute her secret; hence her secret is secure.

Constructing a simulator

To show a particular interactive protocol is zero knowledge, it is necessary to show how to construct Sam for an arbitrary program 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 he 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 he 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 he requests another value from Alice. Pretend that Alice sent him y and continue.
9. Continue simulating Mallory until he 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 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 Mallory generates the b_i's, the chance is very high that Sam's initial guesses were wrong and he will be forced to start over again. Indeed, the probability that all t initial guesses are correct is only $1/2^t$.

Quadratic Residues Revisited

Legendre symbol

Let p be an odd prime, a an integer. The *Legendre symbol* $\left(\frac{a}{p}\right)$ is a number in $\{-1, 0, +1\}$, defined as follows:

$$\left(\frac{a}{p}\right) = \begin{cases} +1 & \text{if } a \text{ is a non-trivial quadratic residue modulo } p \\ 0 & \text{if } a \equiv 0 \pmod{p} \\ -1 & \text{if } a \text{ is } \textit{not} \text{ a quadratic residue modulo } p \end{cases}$$

By the Euler Criterion, we have

Theorem

Let p be an odd prime. Then

$$\left(\frac{a}{p}\right) \equiv a^{\left(\frac{p-1}{2}\right)} \pmod{p}$$

Note that this theorem holds even when $p \mid a$.

Properties of the Legendre symbol

The Legendre symbol satisfies the following *multiplicative property*:

Fact

Let p be an odd prime. Then

$$\left(\frac{a_1 a_2}{p}\right) = \left(\frac{a_1}{p}\right)\left(\frac{a_2}{p}\right)$$

Not surprisingly, if a_1 and a_2 are both non-trivial quadratic residues, then so is $a_1 a_2$. Hence, the fact holds when

$$\left(\frac{a_1}{p}\right) = \left(\frac{a_2}{p}\right) = 1.$$

Product of two non-residues

Suppose $a_1 \notin \mathrm{QR}_p$, $a_2 \notin \mathrm{QR}_p$. The above fact asserts that the product $a_1 a_2$ *is* a quadratic residue since

$$\left(\frac{a_1 a_2}{p}\right) = \left(\frac{a_1}{p}\right)\left(\frac{a_2}{p}\right) = (-1)(-1) = 1.$$

Here's why.

- Let g be a primitive root of p.
- Write $a_1 \equiv g^{k_1} \pmod{p}$ and $a_2 \equiv g^{k_2} \pmod{p}$.
- Both k_1 and k_2 are odd since a_1, $a_2 \notin \mathrm{QR}_p$.
- But then $k_1 + k_2$ is even.
- Hence, $g^{(k_1+k_2)/2}$ is a square root of $a_1 a_2 \equiv g^{k_1+k_2} \pmod{p}$, so $a_1 a_2$ is a quadratic residue.

The Jacobi symbol

The *Jacobi symbol* extends the Legendre symbol to the case where the "denominator" is an arbitrary odd positive number n.

Let n be an odd positive integer with prime factorization $\prod_{i=1}^{k} {p_i}^{e_i}$. We define the *Jacobi symbol* by

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i} \tag{1}$$

The symbol on the left is the Jacobi symbol, and the symbol on the right is the Legendre symbol.

(By convention, this product is 1 when $k = 0$, so $\left(\frac{a}{1}\right) = 1$.)

The Jacobi symbol extends the Legendre symbol since the two definitions coincide when n is an odd prime.

Meaning of Jacobi symbol

What does the Jacobi symbol mean when n is not prime?

- If $\left(\frac{a}{n}\right) = +1$, a might or might not be a quadratic residue.
- If $\left(\frac{a}{n}\right) = 0$, then $\gcd(a, n) \neq 1$.
- If $\left(\frac{a}{n}\right) = -1$ then a is definitely not a quadratic residue.

Jacobi symbol $= +1$ for $n = pq$

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

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right) \tag{2}$$

there are two cases that result in $\left(\frac{a}{n}\right) = 1$:

1. $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = +1$, or
2. $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$.

Case of both Jacobi symbols $= +1$

If $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = +1$, then $a \in \mathrm{QR}_p \cap \mathrm{QR}_q = Q_n^{11}$.

It follows by the Chinese Remainder Theorem that $a \in \mathrm{QR}_n$.

This fact was implicitly used in the proof sketch that $|\sqrt{a}| = 4$.

Case of both Jacobi symbols $= -1$

If $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$, then $a \in \mathrm{QNR}_p \cap \mathrm{QNR}_q = Q_n^{00}$.

In this case, a is *not* a quadratic residue modulo n.

Such numbers a are sometimes called "pseudo-squares" since they have Jacobi symbol 1 but are not quadratic residues.

Computing the Jacobi symbol

The Jacobi symbol $\left(\frac{a}{n}\right)$ is easily computed from its definition (equation 1) and the Euler Criterion, given the factorization of n.

Similarly, $\gcd(u, v)$ is easily computed without resort to the Euclidean algorithm given the factorizations of u and v.

The remarkable fact about the Euclidean algorithm is that it lets us compute $\gcd(u, v)$ efficiently, without knowing the factors of u and v.

A similar algorithm allows us to compute the Jacobi symbol $\left(\frac{a}{n}\right)$ efficiently, without knowing the factorization of a or n.

Identities involving the Jacobi symbol

The algorithm is based on identities satisfied by the Jacobi symbol:

1. $\left(\frac{0}{n}\right) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n \neq 1; \end{cases}$

2. $\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \equiv \pm 1 \pmod 8 \\ -1 & \text{if } n \equiv \pm 3 \pmod 8; \end{cases}$

3. $\left(\frac{a_1}{n}\right) = \left(\frac{a_2}{n}\right)$ if $a_1 \equiv a_2 \pmod n$;

4. $\left(\frac{2a}{n}\right) = \left(\frac{2}{n}\right) \cdot \left(\frac{a}{n}\right)$;

5. $\left(\frac{a}{n}\right) = \begin{cases} \left(\frac{n}{a}\right) & \text{if } a, n \text{ odd and } \neg(a \equiv n \equiv 3 \pmod 4) \\ -\left(\frac{n}{a}\right) & \text{if } a, n \text{ odd and } a \equiv n \equiv 3 \pmod 4. \end{cases}$

A recursive algorithm for computing Jacobi symbol

```
/* Precondition: a, n >= 0; n is odd */
int jacobi(int a, int n) {
  if (a == 0)                          /* identity 1 */
    return (n==1) ? 1 : 0;
  if (a == 2)                          /* identity 2 */
    switch (n%8) {
    case 1: case 7: return 1;
    case 3: case 5: return -1;
    }
  if ( a >= n )                        /* identity 3 */
    return jacobi(a%n, n);
  if (a%2 == 0)                        /* identity 4 */
    return jacobi(2,n)*jacobi(a/2, n);
  /* a is odd */                       /* identity 5 */
  return (a%4 == 3 && n%4 == 3) ? -jacobi(n,a) : jacobi(n,a);
}
```