18.425/6.875 Introduction to Cryptography

Lecturer: Shafi Goldwasser
Scribe: David Wilson
Lecture #21: December 2, 1991

So far we have looked at classical applications of cryptography: encryption, signatures, and pseudorandom number generators. Now we will look at some more recent applications: zero knowledge proofs and other two party protocols. This lecture will deal with two-party protocols. Intuitively a two-party protocol is a method for two mutually untrusting parties to accomplish some task by communicating over a phone line.

Two party protocols (general)

For the rest of the lecture we will assume that there are two participants $A$ and $B$ in the protocols. $A$ and $B$ are probabilistic polynomial time Turing machines that can communicate. There is a public input available to $A$ and $B$ (e.g. a public file and a security parameter $1^k$), as well as inputs private to $A$ and $B$.

Coin flip protocol

The first problem we will look at, that of flipping a coin, is due to Manuel Blum and Silvio Micali. Parties $A$ and $B$, on public input a security parameter $1^k$ and no private inputs, wish to agree on a value $c \in \{0, 1\}$ such that if either participant follows the protocol, $|\Pr(c = 0) - \Pr(c = 1)| < \frac{1}{Q(k)}$ for all polynomials $Q$ for sufficiently large $k$. (the probability is taken over the local internal coin tosses of parties $A$ and $B$.

Claim 1 If there exist one-way permutations, then there exists a coin flip protocol.

Proof: Suppose there is a family of one-way permutations $F = \{f : D_f \rightarrow D_f\}$.

Step 1: Party $A$ chooses $f \in G(1^k)$ and $x \in D_f$. Let $B$ be the hard bit of $f^{-1}$ (i.e. for all probabilistic polynomial time Turing machines $C$ and all polynomials $P \Pr(C(f(x))) = B(f(x))) < \frac{1}{2} + \frac{1}{Q(k)}$, for $k$ large enough. The probability istaken over $x$ such that $-x-=-k$, the choices of $f \in G(1^k)$, and the coin tosses of algorithm $C$). Party $A$ sends to $B f$ and $f(x)$.

Step 2: Party $B$ sends to $A$ a guess $g \in \{0, 1\}$ as to what $B(f(x))$ is.

Step 3: $A$ then sends $x$ to $B$. If $g = B(f(x))$ then the coin is 0, otherwise it is 1. □

Steps 1 and 2 comprise the “commit phase” (in which $A$ essentially commits to $B(f(x))$ by sending $f(x)$) and step 3 is the “reveal phase.”

---

1These notes are based on notes by David Blackston as modified by Shafi Goldwasser, and on Claude Crépeau’s Ph.D. thesis.
Claim 2 If there exist one-way functions then parties $A$ and $B$ can flip coins over a telephone line.

We rely on the following fact which we won’t prove.

Fact 1 If there are one-way functions then there are pseudorandom number generators.

Proof: Let $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$ be a pseudorandom number generator in the public file.

Step 1: Party $B$ picks a random string $r \in \{0,1\}^{3k}$ and sends it to $A$.

Step 2: Party $A$ chooses a bit $b$ and selects a seed $s \in \{0,1\}^k$. If the bit $b$ is 0 then $B$ sends $t = G(s)$. If the bit is 1 then $B$ sends $t = G(s) \oplus r$.

Step 3: Party $B$ sends $A$ a guess $g$ as to what the bit $b$ was. (This finishes the “commit phase.”)

Step 4: Party $B$ sends $A$ the seed $s$. (This is the “reveal phase.”) Party $A$ computes $G(s)$ to find the bit $b$ (i.e if $G(s) = t$, then $b=0$; if $G(s) \oplus r = t$ then $b=1$). If $g = b$ then the coin is 0, otherwise it is 1.

Party $A$ can not distinguish between the output of $G(s)$ and $G(s) \oplus r$ as that would imply that the outcome of $G$ is distinguishable from random strings and $G$ is not a good pseudo random number generator contrary to assumption. For party $B$ to cheat, it would need to find two seeds $s_1$ and $s_2$ so that $G(s_1) = G(s_2) \oplus r$. There are $2^{3k}$ choices for $r = G(s_1) \oplus G(s_2)$, but only $2^{2k}$ choices for pairs of seeds. Thus for a random $r$, the probability that there exist two seeds that allow party $B$ to cheat is $\leq 2^{-k}$. □

Oblivious transfer protocol

We now look at the protocol of oblivious transfer. Party $A$ has a bit $b$. After parties $A$ and $B$ communicate, we want party $B$ to get either $b$ or nothing, each with probability $\frac{1}{2}$. We also want party $A$ to have no idea whether or not $B$ learned $b$. Oblivious transfer seems quite strange, but as we shall see later, the ability to perform oblivious transfer implies the ability to perform any two-party protocol. Namely: Let $g$ be any polynomial time computable function. Let parties $A$ and $B$ have inputs $x$ and $y$ and wish to compute $g(x,y)$ without revealing anything about their inputs that wouldn’t be revealed by $g(x,y)$ itself. (For example $g(x,y) = xvy$ where $x,y$ are boolean variables. For $x = y = 1$, both parties given $g(x,y)$ know the others input, But for $x = 0, y = 1$, party $A$ should not know whether $y = 1$ or $y = 0$).

First we show that the oblivious transfer protocol can be used to implement the 1-out-of-2 oblivious transfer protocol. In this protocol, party $A$ has a two bits $b_0$ and $b_1$, and party $B$ has a bit $c$. At the end of the protocol we want party $B$ to have $b_c$ and have no idea what $b_{\bar{c}}$ is, and party $A$ to have no idea what $c$ is. Using the 1-out-of-2 oblivious transfer protocol, the protocol to compute $f(x,y)$ can be implemented.
Claim 3 If there exists an oblivious transfer protocol, then there exists an 1-out-of-2 oblivious transfer protocol.

Proof: The public file contains $1^k$, a multiple of 3, such that with high probability $B$ will receive at least $k/3$ bits when the oblivious transfer protocol is run $k$ times, but with low probability will $B$ receive as many as $2k/3$ bits.

Step 1: $A$ chooses $k$ random bits $r_1, \ldots, r_k$.

Step 2: $A$ and $B$ run the oblivious transfer protocol with the bits $r_1, \ldots, r_k$.

Step 3: $B$ selects at random $I_0 = \{i_1, \ldots, i_{k/3}\}$ and $I_1 = \{i_{k/3+1}, \ldots, i_{2k/3}\}$ such that $I_0$ and $I_1$ are taken from $1, 2, \ldots, k$, $B$ successfully received bit $r_i$ for $i \in I_0$, and $I_0$ and $I_1$ are disjoint.

Step 4: $B$ sends to $A$ $N_0 = I_c$ and $N_1 = I_{\bar{c}}$.

Step 5: $A$ computes $\hat{b}_x = b_x \oplus \bigoplus_{n \in N_x} r_n$ and sends $\hat{b}_x$ to $B$, for $x = 0, 1$.

Step 6: $B$ computes $b_c = \hat{b}_c \oplus \bigoplus_{i \in I_0} r_i$.

With high probability $B$ knows $r_i$ for each $i \in I_0$ but doesn’t know $r_i$ for some $i \in I_1$. Hence $B$ can compute $b_c$ but not $\bar{b}_c$. Since $A$ has no idea which of the oblivious transfers succeeded, it cannot decide whether $I_0 = N_c$ equals $N_0$ or $N_1$.

Rabin proposed the following method for implementing oblivious transfer.

Step 1: Party $A$ picks a composite number $n$ which is the product of two large primes. $A$ sends to $B$ $n$ and the encrypted version of $b$ $E_n(b)$.

Step 2: $B$ selects an $x \in Z_n^*$ and sends to $A$ $x^2 \mod n$.

Step 3: $A$ computes a square root $z$ of $x^2$ and sends it to $B$. With probability $\frac{1}{2}$, $z \neq \pm x \mod n$, allowing $B$ to factor $n$ and extract $b$. Otherwise $z = \pm x \mod n$ and $B$ won’t be able to factor $n$. In either case $A$ doesn’t know if $z = \pm x \mod n$.

Rabin’s scheme looks like it works, but this isn’t provable. It might be possible for $B$ to cheat. If instead of sending the square of a random element in $Z_n^*$, $B$ sends $x^2 = (n - 1)/2$ or some other special value, knowing only one square root of $x^2$ might be enough to factor $n$. To fix the protocol, $B$ would need to send to $A$ a proof that it knows a square root of $x^2$ without revealing which square root it knows (a zero knowledge proof). Oblivious transfer can be implemented however.

Simultaneous secret exchange protocol

Another two-party protocol is that of simultaneous secret exchange. Parties $A$ and $B$ have some secret data that they wish to exchange “simultaneously.” One possible application is that of certified mail. $A$ has some mail $K_A$ to deliver to $B$. In return $B$ has a receipt $K_B$ to give to $A$. They want the exchange to happen at the same time, so that $A$ doesn’t get a receipt unless $B$ gets the mail, and $B$ doesn’t get the mail unless $A$ gets a receipt.
Here is a suggested protocol for the simultaneous exchange of secret data $s_A$ and $s_B$. Let the common input be $1^k$, $\alpha \in E_{s_A}(s_A)$, $\beta \in E_{s_B}(s_B)$, $n_A$, and $n_B$. The numbers $n_A$ and $n_B$ are each the product of two large primes and are the public keys used to encrypt $s_A$ and $s_B$. Party A’s private input is the factorization of $n_A$ and party B’s private input is the factorization of $n_B$.

Step 1: A picks $a_1, \ldots, a_k$ at random in $Z_{n_B}^*$ and computes $b_i = a_i^2 \mod n_B$. B picks $w_1, \ldots, w_k$ at random in $Z_{n_A}^*$ and computes $x_i = w_i^2 \mod n_A$.

Step 2: A sends all the $b_i$ to B and B sends all the $x_i$ to A.

Step 3: For each $x_i$, A computes $y_i$ and $z_i$ such that $y_i^2 = z_i^2 = x_i \mod n_A$ and $y_i \neq \pm z_i$. For each $b_i$, B computes $c_i$ and $d_i$ with similar restrictions. (Note that either $y_i$ or $z_i$ equals $\pm w_i$ and similarly for $c_i$, $d_i$, and $a_i$.)

Step 4: For $1 \leq j \leq k$, A sends B the $j$th bit of $y_i$ and $z_i$ and B sends A the $j$th bit of $c_i$ and $d_i$ ($1 \leq i \leq k$).

Step 5: A and B figure out the factorizations of $n_B$ and $n_A$. (A computes $\gcd(x_i - y_i, n_B)$ for some $i$ and B computes $\gcd(c_i - d_i, n_A)$ for some $i$. If one party stopped sending the bits early in step 4, then the other party is only one bit behind and at only a factor of 2 disadvantage in factoring the public key.) Once they factor the public keys, A and B can extract the secret information $s_B$ and $s_A$.

The obvious question is why $k$ numbers are used instead of just one. The answer is that if one number were used, then the protocol is vulnerable to cheating. Suppose B only sends one $x$ to A. A could compute $y$ and $z$ and then just send the $j$th bit of $y$ and some junk bit to B on each iteration of step 4. If $y = \pm w$ then B will not notice the cheating until it is too late. Thus A can cheat with a 50% chance of success. Because A must send at least one square root of $x$ (or else B would notice the cheating right away), A has only a 50% chance of successfully cheating. Since $k$ different $x$’s are sent to A, the probability of A successfully cheating in this way is $2^{-k}$.

However, Shamir and Hastad showed that the protocol is susceptible to cheating. If, instead of choosing the $w_i$’s at random, B chooses $w_1$ at random, sets $x_1 = w_1^2 \mod n_A$, and then sets $x_i = x_1/2^{i-1} \mod n_A$, then after one iteration of step 4, B has all the information it needs to factor $n_A$. This protocol fails for essentially the same reason the above oblivious transfer protocol failed: the parties don’t know whether or not the $x_i$ and $b_i$ are the squares of random elements in $Z_{n_A}^*$ and $Z_{n_B}^*$ respectively.

**Interactive proof systems**

The above two problems in the last two protocols could be solved if we could have a mechanism by which B could prove to A that indeed he knows a square root of the square sent in the oblivious transfer protocol, and by which B could prove to A that (and vice versa) that he selected $w_i$’s in $Z_{n_A}^*$ at random and set $x_i = W_i^2 \mod n$ as required in the secret key exchange protocol. Naturally, these proofs must be done without changing any other properties of the protocol, in particular leaking no knowledge about which square
root B knows in the oblivious transfer protocol, or which square roots of the \( x_i \)'s B knows in the secret key exchange protocol.

Such mechanisms are called zero knowledge proofs. A zero knowledge (interactive) proof is one in which the verifier doesn’t learn anything useful other than that the claim is true. Achieving the Zero Knowledge property requires relaxing and enlarging our notion of acceptable proof. These are called interactive proof systems. In an interactive proof system there are two parties, \( P \) which is a prover, and \( V \) which is a probabilistic polynomial time verifier. The prover and the verifier send messages back and forth, with the prover trying to convince the verifier that some claim is true. If the claim is true, then \( V \) is convinced with probability 1, otherwise \( V \) remains unconvinced with high probability. Thus, the verifier may be erroneously convinced of an incorrect statement (differently from classical proofs) but with exponentially small probability.

Interactive proof systems are interesting from a complexity standpoint because the verifier can be convinced of membership in languages such as \( \text{SAT} \) in probabilistic polynomial time. Next class we will exhibit the power of interactive proofs.