CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 19
March 28, 2012

Secret Splitting

Two-key locks

There are many situations in which one wants to grant access to a resource only if a sufficiently large group of agents cooperate.

For example, the office safe of a supermarket might require both the manager's key and the armored car driver's key in order to be opened.

This protects the store against a dishonest manager or armored car driver, and it also prevents an armed robber from coercing the manager into opening the safe.

A similar 2-key system is used for safe deposit boxes in banks.

Two-part secret splitting

We might like to achieve the same properties for cryptographic keys or other secrets.

Let k be the key for a symmetric cryptosystem. One might wish to split k into two *shares* k_1 and k_2 so that by themselves, neither k_1 nor k_2 by itself reveals any information about k, but when suitably combined, k can be recovered.

A simple way to do this is to choose k_1 uniformly at random and then let $k_2 = k \oplus k_1$.

Both k_1 and k_2 are uniformly distributed over the key space and hence give no information about k.

However, combined with XOR, they reveal k, since $k = k_1 \oplus k_2$.

Comparison with one-time pad

Indeed, the one-time pad cryptosystem of Lecture 3 can be viewed as an instance of secret splitting.

Here, Alice's secret is her message m.

The two shares are the ciphertext c and the key k.

Neither by themselves gives any information about m, but together they reveal $m = k \oplus c$.

Multi-part secret splitting

Secret splitting generalizes to more than two shares.

Imagine a large company that restricts access to important company secrets to only its five top executives, say the president, vice-president, treasurer, CEO, and CIO.

They don't want any executive to be able to access the data alone since they are concerned that an executive might be blackmailed into giving confidential data to a competitor.

Multi-part secret splitting (cont.)

On the other hand, they also don't want to require that all five executives get together to access their data because

- this would be cumbersome;
- they worry about the death or incapacitation of any single individual.

They decide as a compromise that any three of them should be able to access the secret data, but one or two of them operating alone should not have access.

Shamir's Secret Splitting Scheme

(τ, k) threshold secret spitting scheme

A (τ, k) *threshold secret splitting scheme* splits a secret s into *shares* $s_1, \ldots, s_k$.

Any subset of τ or more shares allows s to be recovered, but no subset of shares of size less than τ gives any information about s.

The executives of the previous example thus want a $(3, 5)$ threshold secret splitting scheme: The secret key is to be split into 5 shares, any 3 of which allow the secret to be recovered.

A threshold scheme based on polynomials

Shamir proposed a threshold scheme based on polynomials.

A *polynomial of degree d* is an expression

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d,$$

where $a_d \neq 0$.

The numbers $a_0, \ldots, a_d$ are called the *coefficients* of f.

A polynomial can be simultaneously regarded as a function and as an object determined by its vector of coefficients.

Interpolation

Interpolation is the process of finding a polynomial that goes through a given set of points.

Fact

Let $(x_1, y_1), \ldots, (x_k, y_k)$ be points, where all of the x_i's are distinct. There is a unique polynomial $f(x)$ of degree at most $k-1$ that passes through all k points, that is, for which $f(x_i) = y_i$ $(1 \leq 1 \leq k)$.

f can be found using Lagrangian interpolation. This statement generalizes the familiar statement from high school geometry that two points determine a line.

Lagrangian interpolation method

One way to understand Lagrangian interpolation is to consider the polynomial

$$\delta_i(x) = \frac{(x - x_1)(x - x_2)\ldots(x - x_{i-1})\cdot(x - x_{i+1})\ldots(x - x_k)}{(x_i - x_1)(x_i - x_2)\ldots(x_i - x_{i-1})\cdot(x_i - x_{i+1})\ldots(x_i - x_k)}$$

Although this looks at first like a rational function, it is actually just a polynomial in x since the denominator contains only the x-values of the given points and not the variable x.

$\delta_i(x)$ has the easily-checked property that $\delta_i(x_i) = 1$, and $\delta_i(x_j) = 0$ for $j \neq i$.

Lagrangian interpolation method (cont.)

Using $\delta_i(x)$, the polynomial

$$p(x) = \sum_{i=1}^{k} y_i \, \delta_i(x)$$

is the desired interpolating polynomial, since $p(x_i) = y_i$ for $i = 1, \ldots, k$.

To actually find the coefficients of $p(x)$ when written as

$$p(x) = \sum_{i=0}^{k} a_i x^i,$$

it is necessary to expand $p(x)$ by multiplying out the factors and collect like terms.

Interpolation over finite fields

Interpolation also works over finite fields such as $\mathbf{Z}_p$ for prime p.

It is still true that any k points with distinct x coordinates determine a unique polynomial of degree at most $k-1$ over $\mathbf{Z}_p$.

Of course, we must have $k \le p$ since $\mathbf{Z}_p$ has only p distinct coordinate values in all.

Shamir's secret splitting scheme

Here's how Shamir's (τ, k) secret splitting scheme works.

Let Alice (also called the *dealer*) have secret s.

She constructs a polynomial of degree at most $\tau - 1$ as follows:

- She sets $a_0 = s$.
- She chooses $a_1, \ldots, a_{\tau-1} \in Z_p$ at random.
- She chooses $x_i = i$. $\quad (1 \leq i \leq k)$
- She chooses $y_i = f(i)$. $\quad (1 \leq i \leq k)$[1]
- Share $s_i = (x_i, y_i) = (i, f(i))$.

[1] $f(i)$ is the result of evaluating the polynomial f at the value $x = i$. All arithmetic is over the field $\mathbf{Z}_p$, so we omit explicit mention of mod p.

τ shares are necessary and sufficient to reconstruct s

Theorem

s can be reconstructed from any set T of τ or more shares.

Proof.

Suppose $s_{i_1}, \ldots, s_{i_\tau}$ are τ distinct shares in T.

By interpolation, there is a unique polynomial $g(x)$ of degree $d \le \tau - 1$ that passes through these shares.

By construction of the shares, $f(x)$ also passes through these same shares; hence $g = f$ as polynomials.

In particular, $g(0) = f(0) = s$ is the secret. $\square$

Theorem

Any set T' of fewer than τ shares gives no information about s.

Proof.

Let $T' = \{s_{i_1}, \dots, s_{i_r}\}$ be a set of $r < \tau$ shares.

There are in general many polynomials of degree $\leq \tau - 1$ that interpolate the points in T'.

In particular, for each $s' \in \mathbf{Z}_p$, there is a polynomial $g_{s'}$ that interpolates the shares in $T' \cup \{(0, s')\}$.

Each of these polynomials passes through all of the shares in T', so each is a plausible candidate for f. Moreover, $g_{s'}(0) = s'$, so each s' is a plausible candidate for the secret s.

One can show further that the number of polynomials that interpolate $T' \cup \{(0, s')\}$ is the same for each $s' \in \mathbf{Z}_p$, so each possible candidate s' is equally likely to be s.

Hence, the shares in T' give no information at all about s. $\square$

Secret splitting with semi-honest parties

Shamir's scheme is an example of a protocol that works assuming *semi-honest* parties.

These are players that follow the protocol but additionally may collude in an attempt to discover secret information.

We just saw that no coalition of fewer than τ players could learn anything about the dealer's secret, even if they pooled all of their shares.

Secret splitting with dishonest dealer

In practice, either the dealer or some of the players (or both) may be dishonest and fail to follow the protocol. The honest players would like some guarantees even in such situations.

A dishonest dealer can always lie about the true value of her secret. Even so, the honest players want assurance that their shares do in fact encode a unique secret, that is, all sets of τ shares reconstruct the same secret s.

Failure of Shamir's scheme with dishonest dealer

Shamir's (τ, k) threshold scheme assumes that all shares lie on a single polynomial of degree at most $k-1$.

This might not hold if the dealer is dishonest and gives bad shares to some of the players.

The players have no way to discover that they have bad shares until later when they try to reconstruct s.

Verifiable secret sharing

In *verifiable secret sharing*, the sharing phase is an active protocol involving the dealer and all of the players.

At the end of this phase, either the dealer is exposed as being dishonest, or all of the players end up with shares that are consistent with a single secret.

Needless to say, protocols for verifiable secret sharing are quite complicated.

Dishonest players

Dishonest players present another kind of problem. These are players that fail to follow the protocol. During the reconstruction phase, they may fail to supply their share, or they may present a (possibly different) corrupted share to each other player.

With Shamir's scheme, a share that just disappears does not prevent the secret from being reconstructed, as long as enough valid shares remain.

But a player who lies about his share during the reconstruction phase can cause other players to reconstruct incorrect values for the secret.

Fault-tolerance in secret sharing protocols

A *fault-tolerant secret sharing scheme* should allow the secret to be correctly reconstructed, even in the face of a certain number of corrupted shares.

Of course, it may be desirable to have schemes that can tolerate dishonesty in both dealer and a limited number of players.

The interested reader is encouraged to explore the extensive literature on this subject.

Bit Commitment Problem

Bit guessing game

Alice and Bob want to play a guessing game over the internet.

Alice says,

> *"I'm thinking of a bit. If you guess my bit correctly, I'll give you $10. If you guess wrong, you give me $10."*

Bob says,

> *"Ok, I guess zero."*

Alice replies,

> *"Sorry, you lose. I was thinking of one."*

Preventing Alice from changing her mind

While this game may seem fair on the surface, there is nothing to prevent Alice from changing her mind after Bob makes his guess.

Even if Alice and Bob play the game face to face, they still must do something to *commit* Alice to her bit before Bob makes his guess.

For example, Alice might be required to write her bit down on a piece of paper and seal it in an envelope.

After Bob makes his guess, he opens the envelope to know whether he won or lost.

Writing down the bit commits Alice to that bit, even though Bob doesn't learn its value until later.

Bit-commitment problem

A *bit commitment* or *blob* or *cryptographic envelope* is an electronic analog of a sealed envelope.

Intuitively, a blob has two properties:

1. The bit inside the blob remains hidden until the blob is opened.
2. The bit inside the blob cannot be changed, that is, blob cannot be opened in different ways to reveal different bits.

Bit-commitment primitives

A blob is produced by a protocol **commit**(b) between Alice and Bob. We assume initially that only Alice knows b.

At the end of the commit protocol, Bob has a blob c containing Alice's bit b, but he should have no information about b's value.

Later, Alice and Bob can run a protocol **open**(c) to reveal the bit contained in c to Bob.

Requirements for bit commitment

Alice and Bob do not trust each other, so each wants protection from cheating by the other.

- Alice wants to be sure that Bob cannot learn b after running **commit**(b), even if he cheats.
- Bob wants to be sure that all successful runs of **open**(c) reveal the same bit b', no matter what Alice does.

Note that we do *not* require that Alice tell the truth about her private bit b. A dishonest Alice can always pretend her bit was $b' \neq b$ when producing c. But if she does, c can only be opened to b', not to b.

These ideas should become clearer in the protocols below.

Bit Commitment Using Symmetric Cryptography

A naïve approach to building a bit-commitment scheme

A naïve way to use a symmetric cryptosystem for bit commitment is for Alice to encrypt b with a private key k to get blob $c = E_k(b)$.

She opens it by releasing k. Anyone can compute $b = D_k(c)$.

Alice can easily cheat if she can find a *colliding triple* (c, k_0, k_1) with the property that $D_{k_0}(c) = 0$ and $D_{k_1}(c) = 1$.

She "commits" by sending c to Bob.

Later, she can choose to send Bob either k_0 or k_1.

This isn't just a hypothetical problem. Suppose Alice uses the most secure cryptosystem of all, a one-time pad, so $D_k(c) = c \oplus k$.

Then $(c, c \oplus 0, c \oplus 1)$ is a colliding triple.

Another attempt

The protocol below tries to make it harder for Alice to cheat by making it possible for Bob to detect most bad keys.

	Alice		Bob
	To **commit**(b):		
1.		$\xleftarrow{r}$	Choose random string r.
2.	Choose random key k.		
	Compute $c = E_k(r \cdot b)$.	$\xrightarrow{c}$	c is commitment.
	To **open**(c):		
3.	Send k.	$\xrightarrow{k}$	Let $r' \cdot b' = D_k(c)$.
			Check $r' = r$.
			b' is revealed bit.

Security of second attempt

For many cryptosystems (e.g., DES), this protocol does indeed prevent Alice from cheating, for she will have difficulty finding any two keys k_0 and k_1 such that $E_{k_0}(r \cdot 0) = E_{k_1}(r \cdot 1)$, and r is different for each run of the protocol.

However, for the one-time pad, she can cheat as before: She just takes c to be random and lets $k_0 = c \oplus (r \cdot 0)$ and $k_1 = c \oplus (r \cdot 1)$.

Then $D_{k_b}(c) = r \cdot b$ for $b \in \{0, 1\}$, so the revealed bit is 0 or 1 depending on whether Alice sends k_0 or k_1 in step 3.

Need for a different approach

We see that not all secure cryptosystems have the properties we need in order to make the protocol secure.

We need a property analogous to the strong collision-free property for hash functions (Lecture 15).

Bit Commitment Using Hash Functions

Bit commitment from a hash function

The analogy between bit commitment and hash functions described above suggests a bit commitment scheme based on hash functions.

	Alice		Bob
	To **commit**(b):		
1.		$\xleftarrow{r_1}$	Choose random string r_1.
2.	Choose random string r_2.		
	Compute $c = H(r_1 r_2 b)$.	$\xrightarrow{c}$	c is commitment.
	To **open**(c):		
3.	Send r_2.	$\xrightarrow{r_2}$	Find $b' \in \{0, 1\}$ such that $c = H(r_1 r_2 b')$. If no such b', then fail. Otherwise, b' is revealed bit.

Purpose of r_2

The purpose of r_2 is to protect Alice's secret bit b.

To find b before Alice opens the commitment, Bob would have to find r_2' and b' such that $H(r_1 r_2' b') = c$.

This is akin to the problem of inverting H and is likely to be hard, although the one-way property for H is not strong enough to imply this.

On the one hand, if Bob succeeds in finding such r_2' and b', he has indeed inverted H, but he does so only with the help of r_1 — information that is not generally available when attempting to invert H.

Purpose of r_1

The purpose of r_1 is to strengthen the protection that Bob gets from the hash properties of H.

Even without r_1, the strong collision-free property of H would imply that Alice cannot find c, r_2, and r_2' such that $H(r_2 0) = c = H(r_2' 1)$.

But by using r_1, Alice would have to find a new colliding pair for each run of the protocol.

This protects Bob by preventing Alice from exploiting a few colliding pairs for H that she might happen to discover.

Bit Commitment Using Pseudorandom Sequence Generators

Bit commitment using a PRSG

Let $G_\rho(s)$ be the first ρ bits of $G(s)$. (ρ is a security parameter.)

	Alice		Bob
	To **commit**(b):		
1.		$\xleftarrow{r}$	Choose random $r \in \{0,1\}^\rho$.
2.	Choose random seed s.		
	Let $y = G_\rho(s)$.		
	If $b = 0$ let $c = y$.		
	If $b = 1$ let $c = y \oplus r$.	$\xrightarrow{c}$	c is commitment.
	To **open**(c):		
3.	Send s.	$\xrightarrow{s}$	Let $y = G_\rho(s)$.
			If $c = y$ then reveal 0.
			If $c = y \oplus r$ then reveal 1.
			Otherwise, fail.

Security of PRSG bit commitment

Assuming G is cryptographically strong, then c will look random to Bob, regardless of the value of b, so he will be unable to get any information about b.

Why?

Assume Bob has advantage ϵ at guessing b when he can choose x and is given c. Here's a judge J for distinguishing $G(S)$ from U.

- Given input y, J chooses random b and simulates Bob's cheating algorithm. J simulates Bob choosing r, computes $c = y \oplus r^b$, and continues Bob's algorithm to find a guess $\hat{b}$ for b.
- If $\hat{b} = b$, J outputs 1.
- If $\hat{b} \neq b$, J outputs 0.

The judge's advantage

If y is drawn at random from U, then c is uniformly distributed and independent of b, so J outputs 1 with probability $1/2$.

If y comes from $G(S)$, then J outputs 1 with the same probability that Bob can correctly guess b.

Assuming G is cryptographically strong, then Bob has negligible advantage at guessing b.

Purpose of r

The purpose of r is to protect Bob against a cheating Alice.

Alice can cheat if she can find a triple (c, s_0, s_1) such that s_0 opens c to reveal 0 and s_1 opens c to reveal 1.

Such a triple must satisfy the following pair of equations:

$$\left.\begin{array}{rcl} c & = & G_\rho(s_0) \\ c & = & G_\rho(s_1) \oplus r. \end{array}\right\}$$

It is sufficient for her to solve the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1)$$

for s_0 and s_1 and then choose $c = G_\rho(s_0)$.

How big does ρ need to be?

We now count the number of values of r for which the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1)$$

has a solution.

Suppose n is the seed length, so the number of seeds is $\leq 2^n$. Then the right side of the equation can assume at most $2^{2n}/2$ distinct values.

Among the 2^ρ possible values for r, only 2^{2n-1} of them have the possibility of a colliding triple, regardless of whether or not Alice can feasibly find it.

Hence, by choosing ρ sufficiently much larger than $2n - 1$, the probability of Alice cheating can be made arbitrarily small.

For example, if $\rho = 2n + 19$ then her probability of successful cheating is at most 2^{-20}.

Why does Bob need to choose r?

Why can't Alice choose r, or why can't r be fixed to some constant?

If Alice chooses r, then she can easily solve $r = G_\rho(s_0) \oplus G_\rho(s_1)$ and cheat.

If r is fixed to a constant, then if Alice ever finds a colliding triple (c, s_0, s_1), she can fool Bob every time.

While finding such a pair would be difficult if G_ρ were a truly random function, any specific PRSG might have special properties, at least for a few seeds, that would make this possible.

Example

For example, suppose $r = 1^\rho$ and $G_\rho(\neg s_0) = \neg G_\rho(s_0)$ for some s_0.

Then taking $s_1 = \neg s_0$ gives
$G_\rho(s_0) \oplus G_\rho(s_1) = G_\rho(s_0) \oplus G_\rho(\neg s_0) = G_\rho(s_0) \oplus \neg G_\rho(s_0) = 1^\rho = r$.

By having Bob choose r at random, r will be different each time (with very high probability).

A successful cheating Alice would be forced to solve
$r = G_\rho(s_0) \oplus G_\rho(s_1)$ in general, not just for one special case.