YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security *Notes 20 (rev. 1)*
Professor M. J. Fischer *November 16, 2006*

Lecture Notes 20

100 Shamir's Secret Splitting Scheme

Shamir proposed a threshold scheme based on polynomials. A *polynomial* of *degree* d is an expression

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_dx^d. \tag{1}$$

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 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.

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 $j \neq i$. Then 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 $i = 1, \ldots, k$. Note that to actually find the coefficients of $p(x)$ when written in the canonical form of equation 1, it is necessary to expand $p(x)$ by multiplying out the factors and collecting like terms.

Interpolation also works over finite fields, for example, $\mathbf{Z}_p$ for prime p. That is, 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 $k \leq p$ since $\mathbf{Z}_p$ has only p distinct coordinate values in all.

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 $a_0 = s$, and she chooses $a_1, \ldots, a_{\tau-1} \in Z_p$ at random. Share s_i is the point (x_i, y_i), where $x_i = i$ and $y_i = f(i)$ $(1 \leq i \leq k)$[1].

Theorem 1 *s can be reconstructed from any set T of τ or more shares.*

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

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 $d \leq \tau - 1$ 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. ■

Theorem 2 *Any set T' of fewer than τ shares gives no information about s.*

Proof: Let $T' = \{s_{i_1}, \ldots, s_{i_r}\}$ 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. ■

101 Secret Splitting with Dishonest Parties

Several variations on secret sharing have been studied. I mention two briefly but do not go into details.

101.1 Verifiable secret sharing

A *dealer* has a secret s which she wishes to share with a number of *players*. The dealer can of course always lie about the true value of her secret, but, as with bit commitment, the players want assurance that their shares do in fact code a unique secret. That is, whenever sufficiently many shares are assembled to reconstruct the secret, the same secret s is recovered, no matter which shares are used. In Shamir's (τ, k) threshold scheme, this will be true only if all of the shares lie on a single polynomial of degree at most $k - 1$. However, if the dealer is dishonest and gives bad shares to some of the players, the resulting shares might not lie on any polynomial of degree $k - 1$ or smaller. The players have no way to discover this until later when they try to reconstruct s.

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.

101.2 Fault tolerance

Even if the dealer is assumed to be honest, there is still the problem of actively dishonest players. 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 if a player lies about his share and presents a corrupted share, then that share might be used by the other players in reconstructing an incorrect value for the secret. 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 certain number of players. The interested reader is encouraged to explore the extensive literature on this subject.

102 Bit-Commitment Problem

Alice and Bob want to play a 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."

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 and knows whether he has won or lost. The act of writing down the bit *commits* Alice to that bit, even though Bob doesn't learn its value until later.

The *bit-commitment problem* is to implement an electronic form of sealed envelope called a *commitment* or *blob* or *cryptographic envelope*. Intuitively, a blob has two properties: (1) It is not possible to see the bit inside the blob without opening it. (2) It is not possible to change the bit inside the blob, that is, the blob cannot be opened in two different ways to reveal two different bits.

A blob is produced by a protocol $\mathbf{commit}(b)$ between Alice and Bob. We assume that b is initially private to Alice. 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 $\mathbf{open}(c)$ to reveal the bit contained in c.

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 $\mathbf{commit}(b)$, even if he misbehaves during the protocol. Bob wants to be sure that any successful run of $\mathbf{open}(c)$ reveals the same bit b', so 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.

103 Bit Commitment Using Symmetric Cryptography

A naïve way to use a symmetric cryptosystem for bit commitment is for Alice to commit b by encrypting it with a private key k to get a blob $c = E_k(b)$. She later opens it using the decryption function $D_k(c)$. Unfortunately, Alice can easily cheat if she can find a "colliding triple" (c, k_0, k_1) with the properties that $D_{k_0}(c) = 0$ and $D_{k_0}(c) = 1$. She just "commits" by sending c to Bob. Later, she can choose whether to open it to 0 or to 1 by sending Bob 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 (lecture notes 3, section 14), so $D_k(c) = c \oplus k$. Then she can easily find a colliding triple by choosing $k_0 = c$ and $k_1 = c \oplus 1$.

The protocol of Figure 1 tries to make it harder for Alice to cheat by making it possible for Bob to detect most bad keys.

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 $E_{k_0}(r \cdot 0) = E_{k_1}(r \cdot 1)$. However, for the one-time pad cryptosystem, 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.

We see that not all secure cryptosystems have the properties we need in order to make the protocol of Figure 1 secure. We need a property analogous to the strong collision-free property for hash functions (lecture notes 14, section 73).

	Alice		Bob
	To $\mathbf{commit}(b)$:		
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 $\mathbf{open}(c)$:		
3.	Send k.	$\xrightarrow{k}$	Let $r' \cdot b' = D_k(c)$.
			Check $r' = r$.
			b' is revealed bit.

Figure 1: Bit commitment using cryptosystem.

104 Bit-Commitment Using Hash Functions

The analogy between bit commitment and hash functions described above suggests a bit-commitment scheme based on hash functions, as shown in Figure 2.

	Alice		Bob
	To $\mathbf{commit}(b)$:		
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 $\mathbf{open}(c)$:		
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.

Figure 2: Bit commitment using hash function.

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.

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.