Threshold Schemes

An important cipher key K must be protected from (a) accidental or malicious exposure (causing vulnerability) and (b) loss or destruction (causing inaccessibility).

Both problems may be alleviated by the use of shadows in a threshold scheme.

An important special case: We want two people to share a secret key K in such a way that both must cooperate to use K. Give one person a random number r the same length as K. Give the other one $K + r$. K cannot be determined from $K + r$ unless one knows r, but then it is easy: $(K + r) - r = K$.

This generalizes easily to more than two people: With three people, give the first two of them random numbers r and s. Give the third one $K + r + s$. When all three cooperate, they can subtract the first two numbers from the third go form K.

Another variation uses $\oplus$ (XOR) instead of $+$ and $-$.

When $1 \leq t \leq w$ are integers, a (t, w) *threshold scheme* is a system of protecting a key K by breaking it into w *shadows* (pieces) $K_1, \ldots, K_w$ in such a way that (a) it is easy to compute K using knowledge of any t of the shadows K_i, but (b) it is impossible to compute K because of lack of information if one knows any $t - 1$ or fewer of the shadows K_i.

The w shadows are given to w users. Since (at least) t shadows are needed to find K, no group of fewer than t users can conspire to get the key.

At the same time, if a shadow is lost or destroyed, one can still compute K so long as at least t valid shadows remain.

The previous slide showed how to construct a (t, t) threshold scheme.

Lagrange Interpolating Polynomial Threshold Scheme

A. Shamir proposed a (t, w) threshold scheme based on Lagrange interpolating polynomials.

A polynomial of degree $t - 1$ is determined by its values at t distinct values of its argument.

The shadows come from a random polynomial of degree $t - 1$:

$$h(x) = (a_{t-1}x^{t-1} + \cdots + a_1 x + a_0) \bmod p$$

with constant term $a_0 = K$.

All arithmetic is done modulo p, where p is a prime number greater than both K and w. Long keys can be broken into smaller blocks to avoid computing modulo a large prime.

Given $h(x)$, the key K is easily computed by $K = h(0)$.

The w shadows are defined as the value of $h(x)$ at w distinct points. For example, one might let $K_i = h(i)$ for $1 \leq i \leq w$.

Given t shadows $K_{i_1}, \ldots, K_{i_t}$, one may construct $h(x)$ as a Lagrange polynomial

$$h(x) = \sum_{s=1}^{t} K_{i_s} \prod_{\substack{j=1 \\ j \neq s}}^{t} \frac{(x - i_j)}{(i_s - i_j)} \bmod p.$$

Example of the LIP Threshold Scheme

Let $t = 3$, $w = 5$, $p = 13$, $K = 10$ and

$$h(x) = (6x^2 + 7x + 10) \bmod 13$$

with random coefficients 6 and 7.

The five shadows are the values of $h(x)$ at $x = 1, 2, 3, 4, 5$:

$$K_1 = h(1) = (6 + 7 + 10) \bmod 13 = 10$$

$$K_2 = h(2) = (24 + 14 + 10) \bmod 13 = 9$$

$$K_3 = h(3) = (54 + 21 + 10) \bmod 13 = 7$$

$$K_4 = h(4) = (96 + 28 + 10) \bmod 13 = 4$$

$$K_5 = h(5) = (150 + 35 + 10) \bmod 13 = 0$$

We can recover $h(x)$ and $K = h(0)$ from any three of the shadows. For example, using K_1, K_3 and K_5 we have:

$$h(x) = \{10\frac{(x-3)(x-5)}{(1-3)(1-5)}+$$

$$7\frac{(x-1)(x-5)}{(3-1)(3-5)}+$$

$$0\frac{(x-1)(x-3)}{(5-1)(5-3)}\} \bmod 13$$

$$= 10(x-3)(x-5)/8+7(x-1)(x-5)/(-4) \bmod 13$$

$$= 10(x-3)(x-5)5 + 7(x-1)(x-5)3 \bmod 13$$

$$= 50(x^2 - 8x + 15) + 21(x^2 - 6x + 5) \bmod 13$$

$$= 11(x^2 + 5x + 2) + 8(x^2 + 7x + 5) \bmod 13$$

$$= (19x^2 + 111x + 62) \bmod 13 = h(x).$$

DSS and DSA

The Digital Signature Standard uses the Digital Signature Algorithm to sign hash functions. Compare with signing a hash function with RSA.

DSA is a variation of signature schemes of ElGamal and Schnorr.

Notation:

p is a prime of L bits ($2^{L-1} < p < 2^L$), $512 \leq L \leq 1024$, $64|L$.

q is a prime of 160 bits, $q|p-1$.

h is an integer, $1 < h < p-1$, $h^{(p-1)/q} \bmod p > 1$, that is, $h^{(p-1)/q} \not\equiv 1 \pmod{p}$.

$g = h^{(p-1)/q} \bmod p$. (g has order q modulo p.)

p, q, g are a global public key used by several people.

Each user of the DSS chooses a secret private key x in $1 < x < q$ and publishes a public key $y = g^x \bmod p$.

This assumes that discrete logarithms are hard to compute.

Each time a user wants to sign a message M, he chooses a secret random number k in $1 < k < q$ and computes SHA of M, called $H(M)$ below.

Alice signs message M with the pair r, s, where

$r = (g^k \bmod p) \bmod q$, and

$s = [k^{-1}(H(M) + xr)] \bmod q$.

If Bob receives the message M' with signature r', s' from Alice, he verifies her signature by computing:

$w = (s')^{-1} \bmod q$,

$u_1 = [H(M')w] \bmod q$,

$u_2 = (r')w \bmod q$,

$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$,

and making the test "$v = r'$?"

If this equality holds, then Bob accepts that $M' = M$ is a message actually sent to him by Alice.

Note that y is Alice's public key.

Why does the DSA work? That is, assuming that the message and signature are received correctly (so $M' = M$, $r' = r$ and $s' = s$), why should $v = r$?

Theorem: If M is unchanged and really came from Alice, then $v = r$.

Subliminal Channels

Subliminal channels are covert ways that an attacker can send a second message hidden within a normal message. A simple example of this is the bit string formed by the parity of the number of letters in each word of a message. A carefully constructed, innocent sounding message may contain another message hidden in this bit string.

As another example, a message may be hidden in certain bits of a digitized image. Some cryptographic algorithms having subliminal channels are schemes that choose random numbers used just once, such as the Digital Signature Algorithm.

Here is a subliminal channel for the DSA. Alice is a spy who sends many plaintext messages to her contact Bob. Alice's employer reviews these messages to ensure she is not divulging any secrets. All of the messages are signed by the DSA. Alice and Bob secretly agree on a prime p_1, different from any parameter of the DSA. When Alice signs an innocuous message M, she hides a subliminal bit in it. If she wants to send a 1 bit, she tries different random numbers k until the r parameter of the signature is a quadratic residue modulo p_1. For a 0 bit, she makes r a quadratic nonresidue modulo p_1. Since half of the r are quadratic residues and half are quadratic nonresidues, she doesn't have to try many random k to do this.

Bob checks each DSA signature to be sure the message came from Alice. Alice's employer could make the same check and find nothing amiss. Bob would read the subliminal bit in each message by evaluating $r^{(p_1-1)/2} \bmod p_1$ and using Euler's Criterion to determine whether r is a QR or QNR modulo p_1.

In fact, Alice could send several subliminal bits per message.

Suppose Bob and Alice agree on j secret primes $p_1, \ldots, p_j$. Then j subliminal bits could be sent per message, with the i-th bit being 0 or 1 according as r is a quadratic residue or non-residue modulo p_i.

On average, Alice would have to try 2^j values of k to get an r with the required properties, so she can't make j too big or her employer might wonder why her DSA was so slow. She could choose $j = 16$ and send two bytes per message.

An evil implementer of DSA could use the subliminal channel to leak Alice's private key.

Mike sells cheap DSA chips with a 14-bit subliminal channel using fourteen secret primes. When the chip signs a message, the user gives it her private 160-bit key x.

Alice buys a DSA chip from Mike and signs messages with it. The chip breaks the 160-bit key x into 16 ten-bit pieces. It chooses a random 4-bit number e and sends the subliminal message (e, the e-th piece of x).

Mike deduces ten bits of Alice's private key from each signature. After many signatures, he learns many ten-bit pieces of x. When he knows most pieces he can compute the remaining bits by trying all possibilities. Then he can forge Alice's signature on messages.

Even if Alice knew that Mike was stealing DSA private keys this way, she could not prove it unless she knew Mike's 14 secret primes.