Lecture 20 - Secret Sharing, Visual Cryptography, Distributed Signatures

Boaz Barak

December 1, 2005

Key protection In cryptography, security is dependent on the adversary not knowing the secret key. However, how can we ensure this property? to be used, the key needs to be stored somewhere, and an adversary might be able, for example, to break into the machine the key is stored. This is especially crucial for *long term* keys such as signature keys, that may be used for many years.

Secret sharing There is no one complete solution to this problem, but there are several cryptographic techniques to tackle it. One of the nicest ones is the idea of *secret sharing*, originally suggested by Shamir.

The idea is to *split* a secret a (which can be a cryptographic key, but can also be any other piece of information) into n pieces (called *shares*), such that for $t \leq n$ (e.g., $t = n$).

- If an adversary has only $t-1$ out of the n shares, then he has *absolutely no information* about the secret a.
- Given t shares it is possible to completely reconstruct the secret a.

Note that a solution that is definitely *not* secure is to just split the secret to consecutive pieces (i.e., if $a \in \{0,1\}^\ell$ have the i^{th} share be the bits of a in the $(i-1)\frac{n}{\ell}$ to the $i\frac{n}{\ell}-1$ positions) - can you see why?

Somewhat surprisingly, Shamir was able to construct a very efficient such scheme for any n and t without relying on any cryptographic assumptions (that is, obtaining information-theoretic Shannon-like security). Such schemes are called t-out-of-n secret sharing schemes. An n-out-of-n schemes is a scheme where all n shares are needed to reconstruct, and if even one share is missing then there's absolutely no information about the secret.

Shamir's scheme Given a string $a \in \{0,1\}^\ell$ and numbers n, t with $t < n$, we choose $d = t-1$ and do the following: (we assume $n < 2^\ell$, otherwise just increase ℓ)

- Let p be a prime number that is between 2^ℓ and $2^{\ell+1}$. Let $\mathbb{F}$ the field of numbers $0, \ldots, p-1$ with operations done modulu p. We can think of a as representing a number in $\mathbb{F}$. We denote this number by a_0.
- Choose independently at random $a_1, \ldots, a_d \leftarrow_{\mathrm{R}} \mathbb{F}$.
- The numbers $a_0, \ldots, a_d$ define a polynomial $p : \mathbb{F} \to \mathbb{F}$ of degree d in the following way: $p(x) = a_0 + a_1 x + \ldots + a_d x^d$. Note that $p(0) = a_0$. For every i between 1 and n, define $s_i = p(i)$.

- The i^{th} share will be s_i.

Defining security To prove that this is a secure secret sharing scheme, we prove the following two lemmas:

Lemma 1. *For every $a \in \{0,1\}^\ell$, and for every $t-1$ positions $i_1, \ldots, i_{t-1}$, the distribution of $s_{i_1}, \ldots, s_{i_{t-1}}$ is the uniform distribution over $\mathbb{F}^{t-1}$.*

This lemma guarantees the secrecy property — it says that no matter what a is, if an adversary gets only $t-1$ pieces then the distribution it sees is the uniform distribution — a distribution that is independent of a and hence gives no information about it.

Lemma 2. *There is an efficient algorithm that for every t positions $i_1, \ldots, i_t$ can recover a from $s_{i_1}, \ldots, s_{i_t}$.*

This lemma guarantees the reconstruction property.

Proofs We'll actually start with the proof of Lemma 2. We'll prove an even stronger property: that we can reconstruct the entire polynomial $a_0, \ldots, a_d$ from the t values. Recall that d is $t-1$. Thus, the algorithm is given $d+1$ positions $i_1, \ldots, i_{d+1} \in F$ and the value of the polynomial $p(\cdot)$ in these positions and needs to reconstruct the $d+1$ coefficients of the polynomial $p(\cdot)$. This is the well known *polynomial interpolation problem* and there is an efficient algorithm to solve it. (This is a higher degree generalization of the problem where you're given two points and need to find the unique line that passes through these two points.)

The algorithm actually only involves solving linear equations: think of $a_0, \ldots, a_d$ as unknowns. We are given $d+1$ equations of the form

$$a_0 + a_1 i_j + \ldots + a_d (i_j)^d = y_j$$

for $j = 1, \ldots, d+1$. We know i_j (and hence also know $(i_j)^k$ for every k) and are given also $y_j = p(i_j)$. Thus, these are $d+1$ linear equations in $d+1$ variables. They have a unique solution if and only if the determinant of the coefficient matrix is non zero. This is a matrix where the entry at the j^{th} row and k^{th} column is $(i_j)^k$. This is called the Vandermounde matrix and its determinant is known to be $\prod_{j \neq j'} (i_j - i_{j'})$ which is non zero since all the i_j's are distinct.

Thus, we can solve these equations and find $a_0, \ldots, a_d$ and in particular find a from a_0.

Proof of Lemma 1. This actually follows from the proof of the previous lemma.

Let $i_1, \ldots, i_{t-1}$ be the $d = t-1$ positions and define $i_0 = 0$. Let a_0 be any number in $\mathbb{F}$. Define function $g_{a_0} : \mathbb{F}^d \to \mathbb{F}^d$ as follows $g_{a_0}(a_1, \ldots, a_d) = (p(i_1), \ldots, p(i_d))$. We claim that g_{a_0} is a permutation. This will imply the lemma since it means that if $a_1, \ldots, a_d$ are chosen at random then $g_{a_0}(a_1, \ldots, a_d)$ is the uniform distribution (and this is exactly what the adversary sees).

To show $g_{a_0}(\cdot)$ is a permutation, it is enough to show an algorithm that inverts it. However, note that $p(0) = a_0$. Thus, the inverter can simply run the algorithm of the previous lemma on the $d+1$ values $p(0), p(i_1), \ldots, p(i_d)$.

Example 1: PGP key recovery mechanism Pretty Good Privacy is a program to provide encrypted email. When writing such a program one is faced with the dilemma of where to actually store the key. It certainly does not make any sense to store the key in the clear in a file on a Windows (or also Mac or Linux) computer, where it can be easily read by any virus/trojan that comes along.[1] However, we can not expect the user to remember the key either, and we can't assume that they have dedicated hardware (e.g., smartcards) to store the keys. The solution PGP uses is to have the users remember a very long password p and to store in the compute $H(p) \oplus k$ (where k is the key and H is a hash function that we think of as a random oracle).[2] However, the user can forget this long password, and in this case might lose completely all access to his email!

The solution PGP used is the following: the key is shared in a 3-out-of-5 scheme to 5 shares $s_1, ldots, s_5$. The user selects 5 personal questions to which he knows the answers $a_1, \ldots, a_5$. The information $H(a_1) \oplus s_1$, ... , $H(a_5) \oplus s_5$ is stored in the computer.[3] If the user remembers the answers to at least 3 of the questions, he can reconstruct the key.

A simple n-out-of-n scheme There is a different scheme for the special case of n-out-of-n which is even simpler than Shamir's scheme: to share $a \in bits^\ell$ choose $a_1, \ldots, a_n$ at random conditioned on $a_1 \oplus a_n = a$. For example, you can choose $a_1, \ldots, a_{n-1}$ independently at random and choose $a_n = a \oplus a_1 \oplus \cdots a_{n-1}$. It's not hard to show that this is indeed an n-out-of-n scheme.

Application 2: visual cryptography The following is a very cute application of secret sharing, obtained by Naor and Shamir: you can break an image I into, say, two images I_1, I_2 such that neither I_1 or I_2 provides any information about I, but if the two are superimposed one on top of the other than I "pops out".

The idea is the following: each pixel in I will be converted into a 2×2 square in I_1 and I_2. In I_1, we'll choose at random whether the shape of that square will be like this:

X	
	X

or like this

	X
X	

In I_2, if the corresponding pixel in I is white, we'll choose the 2×2 square to have exactly the same pattern as I_1, and if it's black we'll choose it to have the opposite pattern.

Thus, if we superimpose them together, then a white pixel will have one of the two patterns above, while a black pixel will become

X	X
X	X

And thus the image will appear (albeit with white pixels converted to gray).

[1]The fact this does not make any sense does not mean that there are no commercial programs that do this.

[2]Actually the requirement from H is that it will be a variant of a *randomness extractor*. Such functions can be obtained without relying on the random oracle assumption. It might be necessary to protect k with a MAC or something similar in addition to XORing it with the password (I believe PGP does that).

[3]Actually, it is stored on a special purpose key reconstruction server, which might be a good idea if the server is more trusted than the user's PC, to help prevent dictionary/brute force attacks.

I don't know of any practical application (although you could perhaps use this to print sensitive documents in a shared printer environment — note that I_1 is independent of I, and so you can prepare a transparency with I_1 ahead of time, and when you want to send a secret document I to the shared printer, just print I_2 instead).

Threshold signatures. Suppose we want to make sure that a signature key stays secure, by splitting it among, say, 5 servers. Now, how do we actually compute a signature? we can have the 5 servers send their share to one another to reconstruct the secret key, and then use it to sign, but if the adversary is eavesdropping while we're doing this, this can be fatal.

Quite amazingly, it is possible for the servers to jointly compute a signature on a message m using the secret shared key *without reconstructing the secret.* We will present a scheme where ℓ servers can share an RSA signing key such that an adversary that can see the private data of $\ell - 1$ of the servers. You can see Shoup's paper on the web site for such a scheme that works in the general t-out-of-n setting, and is also robust (in the sense we'll talk about soon).

Public key Choose $n = pq$ at random, choose e at random from $\{0, \dots, n-1\}$. Let H a random oracle.

Private key Choose d such that $d = e^{-1} \pmod{\phi(n)}$.

Sharing the private key Choose $d_1, \dots, d_\ell$ at random from $\{0, \dots, n\}$ such that $d_1 + \cdots + d_\ell = d \pmod{\phi(n)}$. The i^{th} server gets d_i.

Computing a signature To sign a message m, compute $x = H(m)$. The i^{th} server broadcast $w_i = x^{d_i}$. The signature is $y = w_1 \cdots w_\ell$.
Note that y is a valid RSA signature: $y = x^d$.

Analysis Let A be an adversary that sees the private data of $\ell-1$ servers. For simplicity of notation assume that these are the servers $1, \dots, \ell - 1$. The adversary gets as input $d_1, \dots, d_{\ell-1}$ which are statistically close to random independent elements in $\{0, ldots, n\}$. It gets to choose messages m and ask the ℓ^{th} server to provide it with $w_\ell = x^{d_\ell}$ (where $x = H(m)$) and at the end it needs to output a new m' and x'^d where $x' = H(m')$.

We prove that A will not succeed by simulating A with an adversary A' that forges the standard hash-and-sign RSA signature. The adversary A' gets n and e, and chooses at random $d_1, \dots, d_{ell-1}$ from $0, \dots, n$ and gives them to the adversary A. When A makes a query m, the adversary A' forwards this query to the signing oracle to get x^d where $x = H(m)$. It then computes $w_\ell = x^d x^{-d_1} \cdots x^{-d_{\ell-1}}$ and gives this to A. This is the same value A sees in its interaction with the ℓ^{th} server and so A' has the same success probability as A.

Robustness This analysis was in the so-called *honest but curious* model, where the adversary only sees the private data of the other servers but can not actually control their behavior. However, in many cases we'll want *robust* protocols that guarantee security even if the adversary can actively control the corrupted servers.

There are general transformation of protocols that are secure in the honest-but-curious setting to robust protocols using zero-knowledge proofs, but these come at a steep price in efficiency. There are protocols (often based on special-purpose zero knowledge proofs for specific languages) that achieve these goals more efficiently. In particular a well known scheme for robust secret sharing is Feldman's *verifiable secret sharing*, while as mentioned above, a simple and attractive robust signature scheme is Shoup's. (There are also other, discrete-log based, robust signature schemes that have the advantage of being dealer-less as explained below.)

Dealer-less protocols Another drawback of this protocol is that it requires a trusted dealer that knows the secret and shares it among the servers. In some situations you might want to ensure that secret was *never* held at one location, and was generated jointly by the parties. There are also protocols achieving this goal.

Proactive security When protecting long term keys, you might worry about whether or not the threshold model makes sense. Suppose that you have 5 servers in different locations. You might be convinced that at no point an adversary can compromise more than, say, 2 of them, but it may very well be the case that over the lifetime of the system the adversary might eventually gain access (at least for a small period of time) to each one of the 5 server. Thus, if the servers use long term shares, then eventually the adversary will learn all the shares. The notion of *proactive security* was invented to obtain protocols that are secure even in this situation. The idea is that every once in a while the servers run together a distributed protocol in which they *refresh* their shares. We can ensure that even if 4 shares suffices to reconstruct the secret, if an adversary saw the two shares of servers 1 and 2 before the refresh, and the shares of servers 3 and 4 after the refresh, then he has no information about it.