Secure Data Management

Lectures will not be recorded :(

Course overview

Why secure data management? Because nowadays data is everywhere. Concern is increasing because of its value, both social and economical (e.g. possibility to monitor citizen), so the data protection is getting more and more attention. By the end of this course, students should have knowledge about the state of the art of access control, search on encrypted data and distribute data structures (like distributed ledgers). There will be groups of 4-5 people to get hands-on experience, choosing projects on the three main topics proposed.

Note: the section on pros and cons has been really well highlighted by the professor.

Security Building Blocks

Algebra recap

Short recap on the main algebraic structure that are assumed to be known. An important basic structure is a group, i.e. a set of elements with a binary operation that you can apply to the elements of that group. Such a collection, together with the operation, is actually a group ($G = (S, \cdot)$) if it respects some properties:

- Closed ($\forall a, b \in S, a \cdot b \in S$);
- Associative ($\forall a, b, c \in S, (a \cdot b) \cdot c = a \cdot (b \cdot c)$);
- Identity element $\exists i \in S \mid \forall a \in S, a \cdot i = i \cdot a = a$;
- Contains inverses $\forall a \in S, \exists b \in S \mid a \cdot b = b \cdot a = i$.

These properties will be used and exploited by encryption schemes.

A cyclic group is generated by a generator, which is a special member such that $G = \{g^1, \dots, g^n\}$ (for a group of order n) . To limit the size of a group, it is wise to use modular algebra. Final note, not all groups are commutative (e.g. subtraction is not commutative). A group $(G, \odot)$ is commutative iff $\forall a, b \in G : a \odot b = b \odot a$

Let's see some examples:

- *Multiplicative group of integers modulo* n : its elements are the primitive residue classes modulo n (i.e. the numbers between 1 and n that are relatively prime to n). The operation is multiplication $\mod n$
  E.g.: $\mathbb{Z}_9^* = 1, 2, 4, 5, 7, 8$ where $4 \cdot 8 = 5$ (because $32 \equiv 5 \mod 9$)
- *Additive group of integers modulo* n: the integers from 0 to $n - 1$. The operation is addition $\mod n$.

  E.g.: $\mathbb{Z}_8 = 0, 1, 2, 3, 4, 5, 6, 7$ where $5 + 6 = 3$ (because $11 \equiv 3 \mod 8$)

To complicate things, we go from groups to fields. A field has two operations instead of one (typically $+$ and $\cdot$) and, in addition to the same properties of groups, there is distributivity and it is also required to have commutativity.

~~see slide 8 for additional information~~

Cryptography recap

The first distinction we make is between deterministic and probabilist encryption: in the former the same plain text and the same key generate the same cipher text, while in the latter this does not hold due to the introduction of randomness. Deterministic encryption is vulnerable to brute force attacks, e.g. using dictionaries, while probabilistic encryption is not. An example of probabilistic encryption can be found in block ciphers, when using CBC mode.

There are also several ways to encrypt. We have symmetric (SKE), asymmetric (PKE), or broadcast encryption (BE, one encrypts, many decrypt and not symmetric). It is mostly used in systems that are inherently not having an interactive communication (e.g. encrypt info on physical carriers, blue ray etc).

Additionally there is also the possibility of adopting key encapsulation: encrypting the message with SKE and encrypting the symmetric key using PKE. The PKE is generally much less efficient than SKE, so when a PKE scheme is needed, often key encapsulation is the best option.

It is always important to model trust in cryptosystems: how much can you trust the involved parties?

- Honest, just following the protocol;
- Honest-but-curios (HBC), trying to learn as such as possible, but following the protocol (about the other parties' secrets);
- Malicious, not following a protocol and trying to do bad things. Its goal can be various, e.g. learning as much as possible, misleading other parties, sabotaging the system etc.;
- Untrusted, similar to HBC. no unanimous definition, typically (but not exclusively!) used for HBC parties (when the authors try to overplay the security features of their work...);
- Collusion, when different parties try to work together against others, obtaining more information than they are actually allowed to have, by combining what they have.

Another very important aspect is proofs, is the protocol really offering the security you are aiming for? Three alternatives models are presented.

- ROM (Random Oracle Model), a way to model hash functions using a Random Oracle, that is a black box giving as output a truly random number for any new input (and the same output for repeated inputs). *As far as I understood, we can use this to prove the effectiveness of hash functions, that ideally should behave like the random oracle.*
- GGM (Generic Group Model), a model in which the attacker only knows one possible encoding of the group(s) involved and not an efficient one, so they have to query and oracle to complete operations.
- STM (Standard Model), the adversary does not have enough time nor computational power to break the scheme (e.g. exhaustive search is too complex). AES keys length are becoming longer to keep using this model, so it really relies on the design of a scheme.

A scheme needs to have a solid mathematical analysis to prove that it is secure. That is, in every case, a theoretical proof (of course, barring implementation mistakes) that a scheme cannot be broken by a certain class of attackers. To complete such an analysis, we need to provide a number of information, such as the actual description of the scheme. the description of the attackers, the model, the various assumptions (e.g. discrete log for Diffie-Hellman), the win condition (when is the system broken?) and finally a proof.

A security analysis needs to provide:

- A precise description of the scheme: the participants, their roles, the amount of trust we have in them, the algorithms they run, and the communication between them
- A precise description of the class of attackers: computational power, available time, role in the protocol, ability to corrupt participants (collusion), the extent to which they follow the protocol

- A precise description of the model
- A precise description of the assumptions: certain mathematical problems are assumed to be very hard to compute
- A precise description of the "win condition": when the security of the scheme is considered broken
- A proof that no attacker can achieve the win condition for the proposed scheme in the given model with the described assumptions

Focusing on the class of attackers, there are several properties involved. For example, deterministic vs probabilistic scenarios, time boundaries (polynomial/exponential/unlimited), possible collusions, role in the schema (server/user/third parties etc.), adaptability.

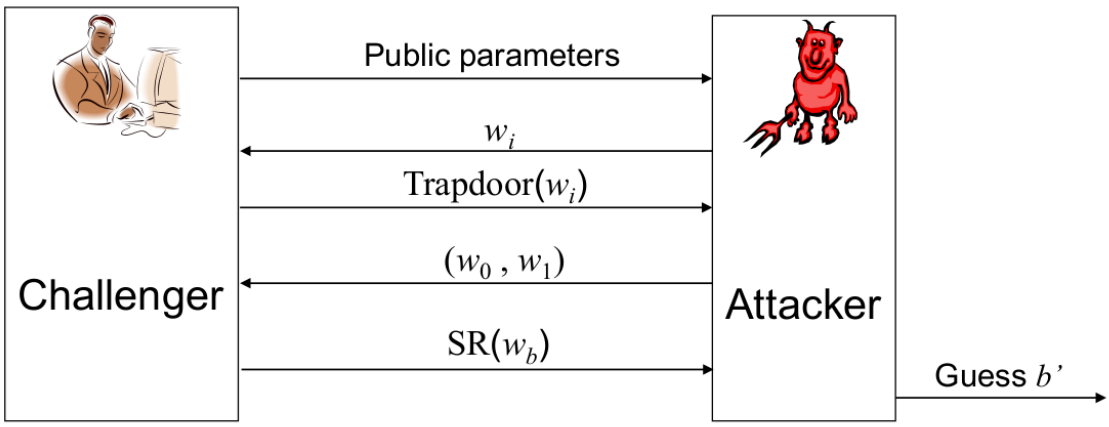
A practical example can be the setting for searchable encryption. Here, he have some methods that are executed in the protocol for its functioning: the plaintext and the master secret key need (of course) to be hidden. To prove the security of such scheme we need to create a model of a polynomially bounded attacker, demonstrating that the generated ciphertext is indistinguishable from random and that trapdoors of other keywords do not reveal any information on the ciphertext. *Here we go into technicalities of this security scheme that however have not been introduced, I think this is done just to show how we can security proof something*.

The different phases of searchable encryption algorithms are the following:

- $Keygen(k)$: the output is the master secret key msk and public parameters $param$
- $Encrypt(param, W, M)$: outputs a ciphertext $S_{W,M}$
- $Trapdoor(W', msk)$: outputs a trapdoor $T_{W'}$
- $Decrypt(T_{W'}, S_{W,M})$: outputs M iff $W = W'$

Security requirement: M and W must be hidden.

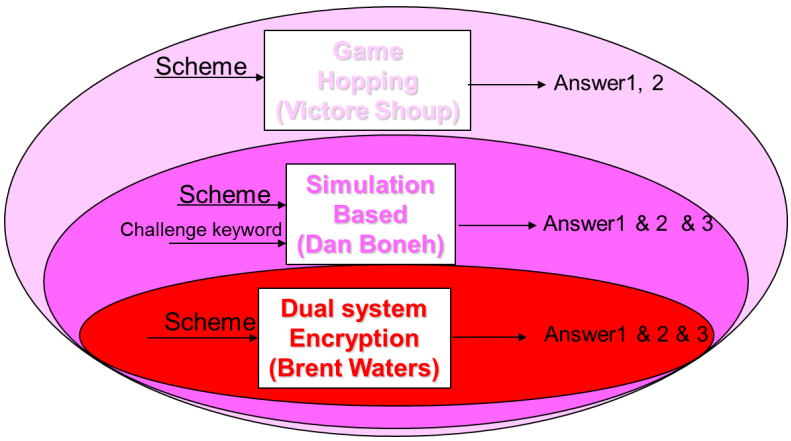
The attacker is allowed to generate any keyword and ask for the trap door. The real attack happens when the attacker understands how the trap door works: he sends two keywords and the challenger is generating one trapdoor for one of those then the attacker can say to which original keyword the trapdoor belongs.



$Pr[b = b'] = 1 / 2 + \varepsilon$. If ε is not negligible attacker wins game and scheme is not secure. if we can prove that ϵ is very small than the attacker is only guessing and he does not really know how trapdoors are constructed.

To prove that is ε negligible security proof must answer these questions

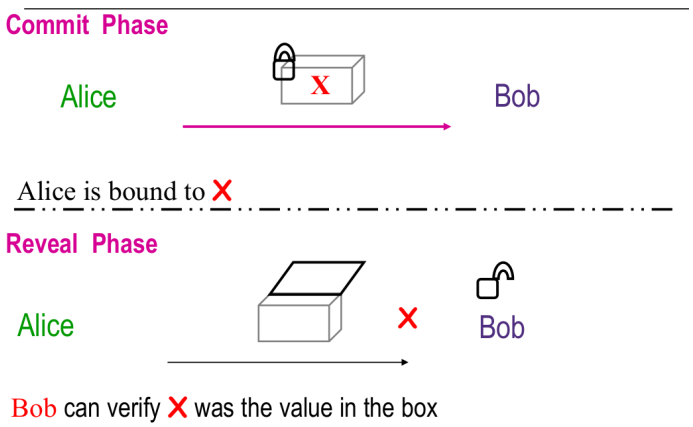
1. Is ciphertext indistinguishable from random?
2. Does trapdoor of keywords other than challenge keyword reveal information on challenge ciphertext?
3. Is trapdoor of challenge keyword simulabile?



~~slides 20-26~~

Commitment Scheme

A commitment scheme is a scheme that allows parties to agree on something remotely. Let's say that Alice and Bob want to toss a coin remotely and based on the result take a decision. It becomes easy to cheat (when the first communicates the result of their toss, the second can decide what to say to win the game). To make this work, Alice would need to commit to her result and this commitment needs to be possible to verify.



Commitment schemes have some security properties.

- **Hiding**: after committing, the receiver does not learn anything about the committed value. There exists two kinds of adversaries: those with infinite computation power and those with limited computation power.
  - **Unconditional** (similar to perfect secrecy): the commitment phase does not leak any information about the committed message, in the information theoretical sense
  - **Computational** (possible only with adversaries with limited computational power): an adversary with limited computation power cannot learn anything about the committed message (similar to semantic security)
- **Binding**: after revealing the commit, the sender cannot successfully change the initial value
  - Unconditional: after the commitment phase, an infinite powerful adversary sender cannot reveal two different values
  - Computational (as above): after the commitment phase, an adversary with limited computation power cannot reveal two different values

This can be done, for example, using the **Pedersen scheme**, leveraging on algebraic properties of a cryptosystem. This solution is unconditionally hiding and computationally binding.

Let g and $h = g^a$ be two generators mod large prime p, picked independently. Commitment to $x : c = g^x h^r$, where r is a random number. To open the commitment the sender sends x and r. The receiver verifies whether $c = g^x h^r$.

- Unconditionally hiding: given a commitment c, every value x is equally likely to be the value committed in c. For example, given x, r, and any x', there exists r' such that $g^x h^r = g^{x'} h^{r'}$, in fact $r' = (x - x')a^{-1} + r \mod q$.
- Computationally binding: suppose the sender open another value $x' \neq x$. That is, the sender find x' and r' such that $c = g^{x'} h^{r'} \mod p$. Now the sender knows x, r, x', and r' s.t., $g^x h^r = g^{x'} h^{r'}$ (mod p), the sender can compute $a = (x' - x) \cdot (r - r') - 1$. Assume DL is hard, the sender cannot open the commitment with another value.

Secret sharing

Secret sharing is all about finding a way to share a secret among the group. Let's say that a company has a secret. If only the CEO knows it, the secret might be lost in case of accidents happening to them. Instead, the secret can be split in many chunks, in such a way that none of the participants can use their chunk to gain knowledge about the secret, but this can be obtained only collaborating with all the other parties (more precisely, there can be a threshold number also smaller than all the participants). We call it a (t, n)-threshold scheme, in which a secret is shared with n parties and at least t need to collaborate to reconstruct it.

You can build a trivial scheme generating $n - 1$ random numbers $s_1, \ldots, s_{n-1}$, then you compute the last number as $s_n = s - \sum_{i=0}^{n-1} s_i$, and you distribute them. However, that is not very solid and $t = n$, which is very often not convenient (in some sense you have the same initial problem of the CEO, but multiplied for the n parties). Shamir Secret Sharing instead relies on polynomials, i.e., building a polynomial $f(x)$ of degree $t - 1$ such that $f(0) = s$ and distributing n points. In this way, with t points and Lagrange interpolation, $f(x)$ can be reconstructed and calculate the secret $s = f(0)$.

~~slides 43-47~~

Lastly, the Blakley schema works in t-dimensional spaces and implies the reconstruction of a secret as intersection point between hyperplanes (which represent the shares).

Verifiable Secret Sharing: there is a reconstructible secret even if the dealer is malicious. How can I know whether a share is correct or not? Note that the correctness of a share can be verified using t other shares.

However, we can't ask other parties to reveal t shares. So each share should have a commitment which is public. The correctness of shares can be verified using commitments. This is called Verifiable Secret Sharing (VSS).

Some examples of such a secret sharing technique are the following:

- Feldman's scheme: based on Shamir's, but commitments to the coefficients of $f(x) = s + a_1 x + a_2 x^2 + \cdots$ are also distributed: $g^s, g^{a_1}, g^{a_2} \ldots$. To verify that your share $(i, f(i))$, where $f(i) = y$, is really a point of the polynomial:

$$g^y \stackrel{?}{=} c_0^{i^0} c_1^{i^1} \ldots c_t^{i^t} = \prod_{j=0}^{t} c_j^{i^j} = \prod_{j=0}^{t} g^{a_j i^j} = g^{\sum_{j=0}^{t} a_j i^j} = g^{f(i)}$$

- Benaloh's scheme: interactive, based on Shamir's, participants can (probabilistically) prove that all the shares are collectively t-consistent (any t shares yield the same polynomial)

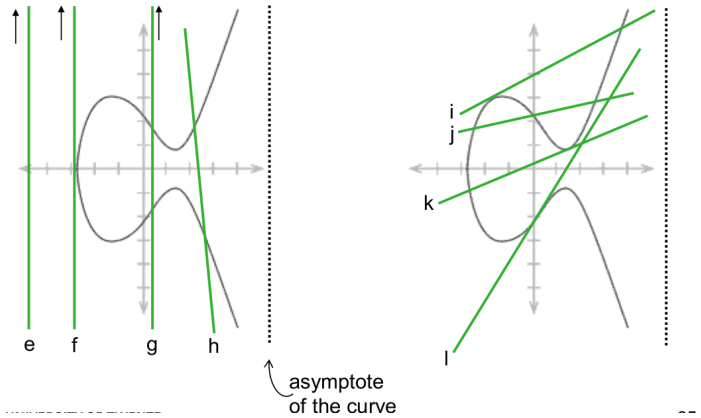
Publicly verifiable secret sharing mean that anybody can verify that the participants received correct shares (e.g.: Chaums and Pedersen Scheme).

~~slides 55 - 61 functional encryption~~

Elliptic Curves

We conclude this lecture talking about elliptic curves, which are the base of modern asymmetric cryptography. EC is a non-singular algebraic curve on the plane and has the formula $y^2 = x^3 + ax + b$ (where $(a, b) \neq (0, 0)$, and a special point O, the point at "infinity") and, thanks to its special properties (*just like Sala says, not going into detail*) it is very difficult for an attacker to reconstruct the private keys generated with this schema. In addition, ECC is much more efficient, hence representing the best solution at the moment.

y is present with only even exponents hence the curve is symmetric to axis x. Every line intersects the curve in 3 points. Exception: vertical lines that only intersect the curve at O, and tangents at inflection points. Tangent points have multiplicity 2.



~~slides 66-68~~