0368.3311 The Theory Behind Blockchains March 6

What Does Security Mean: The Case of Digital Signatures

Lecturer: Nir Bitansky

In the previous lecture, we started looking into what it means for a cryptographic system to be secure. In particular, we specified the class of adversaries that we'd like to protect against; namely, efficient algorithms (formally probabilistic polynomial time). Then we started looking into the case of digital signatures and how to define security.

Before jumping into security, we defined the basic syntax and correctness properties expected from digital signatures.

Definition 0.1 (Signature Scheme)**.** *A signature scheme S consists of polynomial-time algorithms $(Gen, Sign, Ver)$ with the following properties.*

- **Syntax:**
  - *$Gen(1^n)$ is a randomized algorithm takes as input a security parameter n and outputs a secret key sk and a public verification key vk.*
  - *$Sign_{sk}(m)$ is a deterministic algorithm that takes as input a secret key $sk \in \{0,1\}^n$ and message $m \in \{0,1\}^*$, and outputs a signature σ.*
  - *$Ver_{vk}(m, \sigma)$ is a deterministic algorithm that takes as input the verification key vk, message m, and signature σ, and outputs bit (where we shall agree that "1" means accept).*
- **Correctness:** *for any message $m \in \{0,1\}^*$,*

$$\Pr\left[Ver_{vk}(m, Sign_{sk}(m)) = 1 \mid (sk, vk) \leftarrow Gen(1^n)\right] = 1 \ .$$

1 Defining Security of digital Signatures

The first thing about defining security is modeling the type of attacks that we want to defend against; namely, what can the adversary do. In the context of signatures, trying to capture the simple fact that we don't want it to be able to sign messages, unless it knows the secret key, we suggested the following definition.

Definition 1.1 (Unforgeability (first attempt))**.** *For any PPT A and any m^*:*

$$\Pr\left[\ \sigma^* \leftarrow A(vk, m^*), Ver_{vk}(m^*, \sigma^*) = 1 \ \middle|\ (sk, vk) \leftarrow Gen(1^n)\right] \leq \text{negl}(n) \ .$$

This is clearly not enough, as the adversary is likely to also see signatures on certain messages, before trying to forge a signature on a new message. Let's strengthen the definition accordingly.

Definition 1.2 (Unforgeability (second attempt))**.** *For any PPT A and any and polynomial number $k(n) = \text{poly}(n)$ of messages $m_1, \ldots, m_k$:*

$$\Pr\left[\ \sigma^* \leftarrow A(vk, m_1, \sigma_1, \ldots, m_k, \sigma_k), Ver_{vk}(m^*, \sigma^*) = 1 \ \middle|\ \begin{array}{c}(sk, vk) \leftarrow Gen(1^n) \\ \sigma_i \leftarrow Sign_{sk}(m_i)\end{array}\right] \leq \text{negl}(n) \ .$$

Is this enough? what if the adversary is more dynamic — it first sees a bunch of signatures (m_i, σ_i) and then accordingly tries to forge a message $m^* = f(\{m_i\}_i)$ as a function of everything she'd seen so far. The actual security definition of signatures tries to capture such general adversaries.

Definition 1.3 (Unforgeability (actual definition)). *For any oracle aided PPT A:*

$$\Pr\left[\begin{array}{c}(m^*,\sigma^*)\leftarrow A^{Sign_{sk}(\cdot)}(vk)\\ m^*\notin Q, Ver_{vk}(m^*,\sigma^*)=1\end{array}\right]\leq \text{negl}(n)\ ,$$

where $(sk, vk) \leftarrow Gen(1^n)$ *and* Q *is the set of queries that* A *makes to* $Sign_{sk}(\cdot)$.

Remark (What is an oracle aided algorithm?) An oracle-aided algorithm $A^f(x)$ is a standard algorithm that at any point of its execution (and as many times as it wants) can make a query q to the oracle f and get back $f(q)$. (If you want to be formal, you can imagine its a Turing machine that writes queries to a special query tape, and then reads the answer from a special answer tape.)

Going back to the above unforgeability definition, it may seem that we have gone too far. In reality, the adversary does not really obtain access to a signing oracle. Nevertheless, it could be very hard to predict how the adversary can indirectly manipulate signers in order to obtain signatures. This is a recurring issue when defining security of cryptographic primitives. The common paradigm in cryptography is to consider the strongest possible attacker, which will capture all conceivable attacks (now and in the future).

2 How to Construct Signatures that Are Provably Secure?

As we've already noticed, the existence of signatures necessitates that $P \neq NP$, but is this condition enough? What we'd like to show in general is that if we have a hard problem Π, then there exists a corresponding signature scheme $S = S(\Pi)$ so that if Π is hard to solve, then the signature scheme S is hard to break. This is proven by a *reduction* where we take any attacker A against the signature S, and turn it into an algorithm B that solves the problem Π. This is the common paradigm in modern cryptography, often termed *provable security*. It allows for a modular treatment of hardness — once a reduction to a problem is shown we can now algorithmically study this problem to gain confidence in its hardness.

Hard problems. But which problems Π should we use? It would have been great if the (necessary) assumption that $NP \neq P$ would also be enough; namely, if we could base our schemes on an NP complete problem Π, like say $3SAT$. Unfortunately, we do not know how to do it and we typically rely on "more specific problems" like say *the hardness of factoring integers* or *finding discrete logs*.

The common practice in crypto is to try and come up with the simplest possible abstraction of a hard problem that would suffice to construct the desired cryptographic system. For instance, a very simple form of hardness is that of inverting functions.

Definition 2.1 (one-way function). *A (say, injective) function f is a one-way functions if:*

1. *It is easy to compute forward: computing $f(x)$ can be done in polynomial time in $n = |x|$.*
2. *It is hard to invert on random inputs: for any efficient A*

$$\Pr_{x\leftarrow\{0,1\}^n}[A(f(x)) = x] \leq \text{negl}(n)\ .$$

(An analogous definition can be given for functions that are not injective.)

For example:

- The function $f(r)$ that given random coins $r \in \{0,1\}^n$, uses them to first sample two random primes p, q (of length polynomially related to n) and then outputs $N = pq$ is a one-way function assuming that factoring the product of two random primes is hard.
- The discrete log problem, which some of you may have encountered, also gives a OWF.

- OWFs are also necessary for digital signatures; indeed, the function that generates the keys $f(r) = Gen(1^n; r) = (sk, pk)$ is one way (make sure you understand why).
- In fact, OWFs are necessary for almost any goal in cryptography.

Remark (Randomized problems) The hard problems we'll talk about will always be randomized, and we'll always be interested in saying that "they are hard on average" — namely, hard to solve on a random instance. This is different for example from the standard concept of NP hardness where problems should only be hard in "the worst cast", in the sense that any algorithm fails on certain instances of the problem. Hardness on average is quite inherent to cryptography. In a nutshell, this is because we want to be able to choose instances of our hard problem that will be hard for *all* efficient algorithms; indeed we want our systems to be secure against all efficient adversaries.

Example: Signatures from One-Way Functions OWFs turn out to also suffice for signatures.

Theorem 2.2. *[Lam79, GMR84, Gol86, NY89, Rom90] One-way functions are sufficient for digital signature schemes.*

We will not prove this theorem. To give you a taste of how a cryptographic construction and a security proof look like, we will prove a weaker theorem showing only that there exists *one-time signatures*. Then we will only sketch some of the ideas behind general schemes, without giving a proof.

One-Time Signatures. These are schemes where the adversary is allowed to ask for a signature only on a single message of her choice and should then try to forge a signature on a different message.

Definition 2.3 (One-Message Unforgeability)**.** *A signature scheme is one-time unforgeable if for any oracle aided PPT A* **that makes a single oracle query**

$$\Pr\left[\begin{array}{c}(m^*, \sigma^*) \leftarrow A^{Sign_{sk}(\cdot)}(vk) \\ m^* \neq q, Ver_{vk}(m^*, \sigma^*) = 1\end{array}\right] \leq \text{negl}(n) \ ,$$

where $(sk, vk) \leftarrow Gen(1^n)$ and q is the query that A makes to $Sign_{sk}(\cdot)$.

We'll now construct a one-time signature scheme for messages of some fixed length $\ell = \ell(n)$. The construction is as follows:

- $sk = \{x_{i,b} : i \in [\ell], b \in \{0,1\}\}$ where each $x_{i,b} \leftarrow \{0,1\}^n$.
- $vk = \{f(x_{i,b}) : i \in [\ell], b \in \{0,1\}\}$, where f is a OWF.
- $Sign_{sk}(m) = \{x_{i,m_i} : i \in [\ell]\}$
- To verify, apply the function and compare to the verification key.

Claim 2.4. *The above is a secure one-time signature.*

Proof Sketch. Correctness follows easily, we will want to prove security. We want to construct a reduction that takes any adversary that breaks the signatures scheme and turns it into an adversary that breaks the one-way function.

Let A be an adversary that breaks the scheme with noticeable probability $\delta(n) = n^{-\Omega(1)}$, we'll construct an adversary B (with polynomially related running time) that inverts f with probability $\delta/2n$, which is also noticeable and will thus contradict the hardness of inverting f.

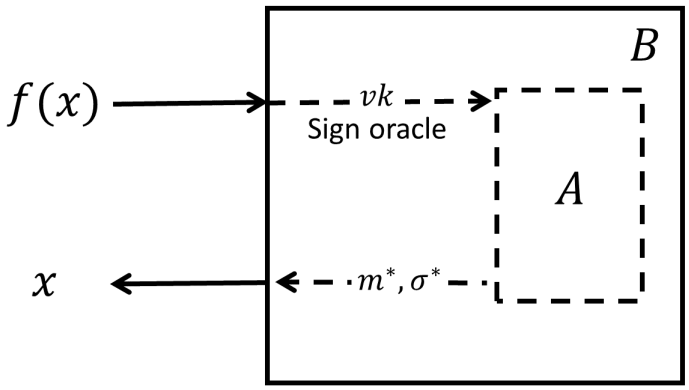

Figure 1: A security reduction. We show how to turn (efficient) attacker A into an (efficient) algorithm B that can invert the one-way function f. B has to somehow embed the challege $y = f(x)$ in a signature game, so that when A will forge, B will be able to extract from this forgery a preimage x with high probability.

$B(y)$**:**

- Samples keys sk, vk as in the actual scheme.
- Samples i, b, and replaces $f(x_{i,b})$ with y, to obtain a new secret key vk'.
- It now emulates $A(vk')$ as follows:
  - When A makes a query m to be signed, if $m_i \neq b$, then B can sign using the secret key sk. If $m_i = b$ it aborts.
  - When A outputs a forgery $m^*, \left\{x^*_{j,m^*_j} : j \in [\ell]\right\}$, if $m^*_i = b$, B outputs x^*_{i,m^*_i} as a preimage for y; otherwise, it aborts.

We claim that B inverts with probability at least $\delta/2\ell$. First, note that whenever A produces a valid forgery, then x^*_{i,m^*_i} is a preimage of y. We next analyze the probability that a valid forgery occurs.

Imagine a tweaked version A' of the adversary A for the signature scheme. A' behaves like A, but in addition samples i, b at random, and if $m_i = b$ or $m^*_i \neq b$ aborts (the moment that the first happens).

Claim 2.5.

1. *A' successfully forges a signature with probability $\delta/2\ell$.*
2. *A forges exactly with the same probability when emulated by A' and when emulated by B.*

To see the first, note that whenever A successfully forges the query m and forgery message m^* are different and thus for some j, $m_j \neq m^*_j$. The probability that i, b chosen by A' coincide with j, m^*_j is $1/2\ell$. In this case, A' succeeds. Overall, it forges with probability at least $\delta/2\ell$.

To see the second, note that in both attacks the emulated attacker A sees exactly the same distribution. □

3 Which Signatures Are Actually Used in the Real World?

While theoretically speaking we can construct signature schemes from the fairly weak and necessary assumption of OWFs, *these are not the schemes that are used in the real world.* In the real world, efficiency and simplicity often dominate. In fact, quite often to increase efficiency/simplicity, real world constructions rely on heuristics and we do not know how to formally reduce their security to a hard problem.

The random oracle model. One very common heuristic, which we will encounter quite often in this course, is *the random oracle heuristic*. Here when we construct schemes and analyze security, we assume that all parties, including the adversary have oracle access to a function $R : \{0,1\}^n \to \{0,1\}^m$ chosen at random.

Having access to such a function turns out to be quite helpful when designing cryptographic systems. For example:

- R is one way.
- R is collision resistant (relevant when $m < n$).
- (R can be used as a pseudorandom function $PRF_{sk}(x) = R(sk, x)$ in the signature tree construction we've mentioned before).
- In the recitation you will see a simple construction of a signature scheme based on a random oracle and the hardness of factoring (or RSA).
- Jumping ahead, we'll use random oracles later on to construct proofs of work and other beasts.

Do random oracles exist? In reality, there are no real random oracles. In fact, a true random oracle will have a huge description. In reality, we need to instantiate such random oracles with a concrete function. A common heuristic is to replace it with a complicated hash function such as SHA-256, which is completely deterministic. Once we do so, however, the security proof, which was established given a truly random oracle, no longer applies — it's a leap of faith. We will not further dwell into this now, but will come back to this point several times during the course.

References

[GMR84] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical"'solution to the signature problem (abstract). In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, page 467, 1984.

[Gol86] Oded Goldreich. Two remarks concerning the goldwasser-micali-rivest signature scheme. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 104–110, 1986.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. Technical report, October 1979.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394, 1990.