0368.3311 The Theory Behind Blockchains 27 February

Computational Concepts and Digital Signatures

Lecturer: Nir Bitansky

1 Efficient Adversaries and Computationally Hard Problems

The theme of cryptography is to replace the need for *trusted parties* with cryptographic protocols whose security is based on computational hardness assumptions. Indeed, most of the goals we'll encounter in this course are outright impossible computationally hard problems do not exist (say, $P = NP$). (We've already seen two such examples: collision-resistant hash functions and digital signatures.) Accordingly, when defining what it means "to be secure" and when constructing protocols we will always aim to deal with computationally bounded adversaries. In what follows, we will get more concrete regarding what we mean by security against computationally-bounded (a.k.a efficient) adversaries.

(t, ε)-Attacks. When addressing an *attack* against a cryptographic system, for instance forging a digital signatures, we will treat two basic measures:

1. The time t (or number of computational steps) required for an attacker algorithm to execute the attack.
2. The probability ε that it succeeds over any randomness used by the system (e.g., when sampling signature keys) and any randomness used by the attacker algorithm.

Indeed, in all cryptographic systems we'll see, it will always be possible to break the scheme if one runs for sufficiently long time, or with small probability. For instance, it will always be possible to break a signature scheme where secret keys sk are of length n, by either exhaustively searching for the key in time 2^n, or by guessing the key w.p. 2^{-n}.

(t, ε)-Security. Naturally, we say that a system is (t, ε)-secure if (t, ε) attacks cannot break it. Typically cryptographic systems will be associated with a certain security parameter n that can be controlled. This security parameter is usually associated with the length of secret keys (or other randomness) and allows to scale the hardness of the scheme (by increasing it we'll get better security). We will accordingly think of $t(n), \varepsilon(n)$ as functions of n.

The Asymptotic Approach. In practice, the exact security of a cryptographic system matters a lot. Due to efficiency considerations, the length n of keys is chosen so that $t(n), \varepsilon^{-1}(n)$ are just above what is considered to be feasible in the actual world (and every once in a while n is updated).

Dealing with concrete security parameters could be quite a hairy business. Throughout the course we'll typically take a more liberal (or rather, theory driven) approach and identify feasible (or efficient) with polynomial and inefficient with superpolynomial. We will use the following terminology:

- We say that a function $f(n)$ is polynomially-bounded if for some constant c, and all $n \in \mathbb{N}$, $f(n) \leq n^c$. We sometimes denote this by $f(n) = n^{O(1)}$.
- We will say that an algorithm A is polynomial-time if its running time is polynomially-bounded (as a function of its input size). (Sometimes, we will say that A is $n^{O(1)}$-time.) More generally, we will consider probabilistic poly-time (PPT) algorithms.
- We will say that a function $\text{negl}(n) \in [0, 1]$ is negligible if it decays faster than any polynomial. That is, for any constant c, there exists n_c, such that for all $n > n_c$ it holds that $\text{negl}(n) \leq n^{-c}$. (Sometimes, we will denote this by $\text{negl}(n) = n^{-\omega(1)}$.)

These definitions behave quite nicely and are thus easy to work with. For instance, $n^{O(1)} \cdot n^{O(1)} = n^{O(1)}$, $\left(n^{O(1)}\right)^{O(1)} = n^{O(1)}$, and $n^{O(1)} \cdot n^{-\omega(1)} = n^{-\omega(1)}$.

2 Understanding Cryptographic Security

We will generally use the above terminology to capture what it means for a cryptographic system to be secure. We will exemplify this through digital signature schemes.

Recall that a signature scheme should allow the holder of the secret signing key to sign any message so that it can be verified by anyone that holds the public verification key. Yet we want that parties that do not know the secret key won't be able to forge signatures; namely, generate signatures on messages for which they haven't already seen a signature. We will now define formally what this means.

Definition 2.1 (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 $m \in \{0,1\}^*$,*

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

- **Unforgeability (first attempt):** *For any PPT A and any m^*:*

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

Is the above unforgeability definition strong enough? does it capture concievable attacks in real life?