CS 4501-6501 Topics in Cryptography 30 Mar 2018

Lecture 10, Zero Knowledge Proofs, Secure Computation

Lecturer: Mahmoody *Scribe: Bella Vice-Van Heyde, Derrick Blakely, Bobby Andris*

1 Introduction

Last time, we discussed RSA encryption and started talking about digital signatures. This lecture, we finished up our discussion on digital signatures and discussed zero knowledge proofs.

2 Reviewing Features of Digital Signatures

Relating Message Authentication Codes and Digital Signatures Digital signature schemes and message authentication codes are both used to ensure the secure authentication of transmitted messages. While message authentication codes do so in a private-key setting, digital signature schemes are the public-key parallel, ensuring secure authentication in a public key setting. Their syntax and security guarantees are analogous, but digital signatures allow us to do more. Digital signatures eliminate the need to share distinct secret keys between the sender and each potential receiver, allowing the sender to publish only one *verification key* that can be used for all subsequent signatures and they all can be publicly verified by all recipients of the messages and their attached signatures.

As in the case of private-key vs. public-key encryption, message authentication codes are shorter and more efficient than digital signatures. Thus, if the sender primarily communicates with only one recipient it is better to share a secret key and use message authentication codes for subsequent authentications.

2.1 Defining Digital Signatures

A digital signature scheme consists of three algorithms $(\mathrm{Gen}, \mathrm{Sign}, \mathrm{Verify})$ such that:

1. The key-generation algorithm $\mathrm{Gen}(1^n)$ is randomized, and also takes the security parameter n, and it outputs a pair of keys (sk, vk). These are called the signing key and the verification key, respectively.

2. The signing algorithm Sign takes as input a signing key and a message m. It outputs a signature σ, namely, $\sigma = \mathrm{Sign}_{sk}(m)$

3. The verification algorithm Verify takes as input a verification key vk, a message m, and a signature σ. If $\mathrm{Verify}_{vk}(m, \sigma) = 1$ we call σ a valid signature on a message m (with respect to some public key vk understood from the context).

Example 2.1. Suppose Alice wants to send a message that can be received by more than one recipient while only sharing one key. She runs $\text{Gen}(1^n)$ to obtain the pair of keys (sk, vk). She publicizes the verification key vk as belonging to her and keeps the signing key sk private. When Alice wants to authenticate a message she uses her signing key sk, signing message m with $\sigma = \text{Sign}_{sk}(m)$. If anyone that has access to this verification key, including Bob, goes over message and signature they can verify whether or not Alice actually sent this message.

Security of Signature Schemes (Informal) Even if Adversary gets to see many signatures correctly signed by Alice, Adversary should not be able to sign any new message m that seems like a valid signature on behalf of Alice. If the Adversary is able to accomplish this we call this a $forgery$. In order to define the security of a signature scheme more formally, a security experiment must first be formalized (similarly to security formalization of MAC).

Signature Scheme Security Game (experiment) Let $\Pi = (\text{Gen}, \text{Sign}, \text{Verify})$ be a signature scheme. Consider the following experiment $\text{SignForge}_{\mathcal{A},\Pi}(n)$ for Adversary $\mathcal{A}$:

1. $\text{Gen}(1^n)$ is run and sk, vk keys are generated
2. Adversary $\mathcal{A}$ is given vk and access to an oracle $\text{Sign}_{sk}(\cdot)$. The adversary then outputs (m, σ). Let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ asked its oracle
3. $\mathcal{A}$ succeeds if and only if (1) $\text{Verify}_{vk}(m, \sigma) = 1$ and (2) $m \notin \mathcal{Q}$. In that case the output of the experiment is defined to be 1.

Definition 2.2 (Formal Definition of Signature Security)**.** : A *signature scheme* $\Pi = (\text{Gen}, \text{Sign}, \text{Verify})$ is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function $negl$ such that:

$$\Pr[\text{SignForge}_{\mathcal{A},\Pi}(n) = 1] \leq negl(n).$$

2.2 Using Trapdoor Permutations for Digital Signatures

When using trapdoor permutations to do the signing job in a signature scheme, there are naive implementations that cause the scheme to not be secure.

1st Idea based on TDPs: Use the public permutation π for the public job of verification using the verification key vk and keep the private trapdoor permutation π^{-1} for the private task of signing using the signing key sk. It intuitively makes sense to use π^{-1} for the signing key because the sender is the only one who should know this information.

Naive implementation of TDPs for signing :

1. The keys are generated using the generation algorithm for TDPs that generates a pair of permutation (π) and its inverse (π^{-1} – or the trapdoor). Let the verification key: "public key" vk be $\pi(\cdot)$ (description of the permutation) and the signing key: "private key" sk be $\pi^{-1}(\cdot)$ (description for the trapdoor).

2. To sign a message m, publish signature $\pi^{-1}(m) = t$ where t is the tag/signature.

3. To verify the message and tag (m, t), accept if and only if: $\pi(t) = m$.

The security issue with TDPs for signing: According to definition 2.2 this encryption scheme is not secure. It allows the adversary to generate a forgery using the public key alone, without obtaining any signatures from the legitimate signer. Namely, the adversary can choose any value for t in the range of the permutation and then generate a corresponding message m by using the public verification key, letting $m = \pi(t)$. This passes the verification test since t is a valid signature on m, and is also a forgery since no signatures at all were issued by the owner of the public key. The argument may be made that this is not a "realistic" attack since the message will most likely not contain any meaningful content, due to the fact that the adversary has no control over the message for which it forges a valid signature. Despite the claim this may not happen in the real world, it is still bad practice to use a scheme that directly goes against a formal security definition. An in fact, in specific contexts, this *might* be indeed a very realistic attack.

3 Hash and Sign Using Ideal Hash Function

The problem created by naively using TDPs for digital signatures is that an adversary can find a message given a tag. They can do this by simply computing $\pi(t)$. We can fix this problem by adding a step to the encryption. First, we hash the message, and then we apply the trapdoor permutation to the hashed message. With this encryption scheme, an adversary can still compute $\pi(t)$, but this will return the value of the hashed message. Assuming an ideal hash function, an adversary would not be able to learn the message from this value.

Formal Description of Hash and Sign Scheme Suppose we have an ideal hash function $h : \{0,1\}^* \rightarrow \{0,1\}^n$ for security parameter n. We also have trapdoor permutation π, π^{-1} on domain $\{0,1\}^n$. Our signing key is π^{-1}, and our verification key is π. To sign m, we first compute $s = h(m)$ and then we output tag $t = \pi^{-1}(s)$. We then verify by checking Verify$(m,t) : h(m) = \pi(t) \rightarrow$ output 1, $h(m) \neq \pi(t) \rightarrow$ output 0.

Theorem If (π, π^{-1}) is a secure TDP, and if h is a random function and if A is an adversary who runs in $poly(n)$, then Pr[winning in "hash and sign"] $\leq negl(n)$.

Proof idea: Assume an adversary A exists that has Pr[winning in "hash and sign"] $>$ $negl(n)$. Turn A into adversary B that breaks the TDP (π, π^{-1}).

4 Zero Knowledge Proofs

Zero knowledge proofs are seemingly magical at first, but they exist! They allow one party (prover) to prove to another party (verifier) that some statements are true, without revealing the actual proof. So, the only thing that is revealed is the truth of the statement, without revealing anything more! How can it be done? Well, we will use an extended notion of proof, known as *interactive* proofs that allows for such surprising phenomenon. Also, as it was shown by Goldreich-Micali-Wigderson, it is enough to do this for *one* NP-complete problem, because having "efficiently verifiable proofs" means having an NP language, and NP languages are all "reducible" to NP complete languages. We start by seeing one Zero Knowledge proof for an NP complete language.

Zero knowledge proofs are also very useful in cryptography. They allow one party to prove to others that they are following the protocol honestly without revealing any other information to them. We will talk about this next time. In fact, the intuition behind the very *definition* of zero knowledge, is the building block of how we define security in many different scenarios, in particular in "secure multi party computation" which is the topic of next time.

4.1 Zero Knowledge Proof for 3-Colorability

3-Colorability This is the problem of determining if it is possible to color a graph using three colors such that no two adjacent nodes are the same color. Describing the problem formally: M is the adjacency matrix for G, so if nodes i and j are adjacent, then $M(i,j) = 1$. And c is a mapping of the set of nodes $\{1, 2, ..., n\}$ to the set of colors (encoded as $\{1, 2, 3\}$). If G is 3-colorable, we are claiming:

$$\exists c : \{1, 2, ..., n\} \to \{1, 2, 3\} \text{ such that } \forall i, j \text{ we have } M(i,j) = 1 \to c(i) \neq c(j)$$

In terms of complexity theory, if the above claim holds then we say G belongs to the language L, the set of all 3-colorable graphs.

Stating the Problem Suppose Alice (the prover) has a graph G and wants to convince Bob (the verifier) that it is 3-colorable without revealing the coloring. How can she prove this to Bob zero knowledge? As in the cave example (see the class video for this), the proof has to be an "interactive proof" where Alice and Bob adhere to some protocol consisting of multiple steps. And this protocol needs to meet two conditions:

1. Soundness, which intuitively means that if Alice is lying, she is very likely to be caught.
2. Zero Knowledge, which means Bob learns nothing about the coloring from his interactions with Alice.

Note that it is crucial for Alice and Bob to interact by sending messages back and forth. As we saw in the case of key agreement and public key cryptography, allowing parties to follow a multi-message protocol allows us to create more complicated systems (zero knowledge proofs are not possible with just one message).

Digital Lockable Envelopes To persuade Bob she knows a 3-coloring of G, Alice needs to somehow share information about the how she colored each node with Bob, which she can send Bob via a set of "digital lockable envelopes," which have two properties. One is that they are essentially encrypted: Bob should not be able to know the contents of the message without asking Alice to unlock it (otherwise she would reveal her information and it would not be zero knowledge). Second, Alice must commit to what she is sending Bob and cannot change the contents of the envelope once she has sealed it. These "digital lockable envelopes" exist as a theoretical construct for discussing zero knowledge proofs. In practice, their function can be achieved via hash functions, by computing the hash of the message concatenated with some randomness (which serves as the "password" for opening the locked envelope). In fact, digital lockable envelopes have their own name and identity in cryptography, they are called *commitment schemes*. We first describe the protocol using the assumption that commitment schemes exist, then we will describe how to do them using hash functions.

First Attempt at a Protocol

1. First Alice puts all the colors of the nodes inside separate lockable envelopes $L_1, \ldots, L_n$ and sends them to Bob.
2. Bob queries randomly queries an edge e from the set $E = \{e_1, e_2, ..., e_m\}$ and asks Alice to open the envelopes containing the colors of the two endpoint i, j nodes.
3. Alice sends the "password" for opening the content of the lockable envelopes L_i, L_j to convince Bob that they indeed have different colors.
4. Bob makes sure that the colors c_i, c_j are both in $\{1, 2, 3\}$ and that they are different.

If Alice does not have a valid coloring of the G, the probability that the endpoints of e are the same color is at least $1/m$. Therefore,

$$\Pr[\textit{Not catching Alice}] \leq 1 - \tfrac{1}{m}$$

To decrease the error and ensure catching a dishonest prover we can repeat the process $n \cdot m$ times, and the error of *not catching* a dishonest prover goes to at most $((1 - 1/m)^m)^n \leq (1/\mathbf{e})^n < 2^{-n}$ (where $\mathbf{e}$ is the natural logarithm's basis logarithm).

However, this is not zero knowledge. Bob clearly learns a tiny bit of information about the coloring from this query and, for example, if he queries n times, he can recover the entire coloring.

Randomization Trick To ensure the proof is zero knowledge, Alice can permute her coloring of G each time she commits a color to an envelope. In other words, she shuffles all of the colors around (e.g., turns red into green, green into blue, and blue into red) before placing a node in an envelope. If Bob then tries to query every edge in the G, he will always realize the endpoints do not match, but because each query was on a different permutation of Alice's coloring of G, if he tries to put everything together, the colors will not match. In fact, he could obtain the exact same information by running a program on his computer that simply generates pairs of numbers (x, y) such that $x, y \in \{1, 2, 3\}$ and $x \neq y$. From his perspective, the information Alice is sending him looks exactly like the output of this program: both are simply a distribution of random pairs of numbers.

Proving Zero Knowledge by means of a simulator In fact, we can *prove* that Bob learned nothing from the interaction by Alice other than the fact that the graph is perhaps 3 colorable. The reason is that if Bob assumes the graph is 3 colorable, he can run a polynomial time algorithm on his side that generates the same information that Bob sees in the interaction with Alice. In particular, a simulator can work as follows. Sim would first pick the random edge $e = (i, j)$ (this is essentially, how the simulator can "cheat" by generating the information in different order than they appear in transaction with Alice, so the simulator's answer is not "convincing" but is carries the same joint distribution). Then Sim would put two different random colors $c_i \neq c_j$ in lockable envelopes E_i, E_j, and then it puts $n - 2$ random numbers for other colors in L_k for $k \neq i, j$. Then, it also puts the password of E_i, E_j as part of the transcript. So, all the generated locks, the color of c_i, c_j and the "password" to locks L_i, L_j are what the simulator outputs. It is easy to see that this is exactly the same distribution of information that the verifier observes in its interaction with the prover. So, everything that verifier learns from talking to the prover can be learned from the simulator as well!

Defining Zero Knowledge We did something unusual above! We proved zero-knowledge without defining it first! This is technically not correct. We cannot have a "proof" without a definition, ever! The more formal approach is that we use the notion of simulators to actually define ZK, and then the existence of the above specific simulator proves that the 3 coloring protocol that we saw is indeed zero knowledge.