University of Illinois, Urbana Champaign
CS/ECE 498AC3/4 Applied Cryptography

Instructor: Dakshita Khurana
Scribe: Vijayendra Jagtap & Kuilin Li
Date: October 6, 2020

LECTURE
13

Schnorr Signatures, Commitments

In previous lecture we developed the concept of digital signatures, which consists of three algorithms: key generation, message signing and message verification. While this schema functions similar to MAC Authentication process, digital signature make use of signing key, sk, in signing process and verification key, pk, in Message Verification process. Digital signatures are widely used to authenticate software updates, authenticate email using domain keys identified mail (DKIM), and creating certificates to authenticate users using certification process.
In this lecture, we move forward and take a look at a new protocol, **Schnorr's Signature**, which achieves security against adversary attacks under the **discrete log assumption**. We will see why this protocol is secure and can be used to sign a message my multiple signers. Lastly, we will end this lecture by discussing **Commitment Scheme**, that allows one party to commit to a chosen value (or chosen message) while keeping it hidden to public, with the ability to reveal the committed value at a later time.

13.1 Schnorr's Identification Protocol

Schnorr's Identification was developed by Claus Schnorr and this protocol is known for its simplicity as well as its intractability of certain discrete log problems. Lets say Alice wants to convince Bob that she knows a message, m. The easiest way to achieve this is by using H(m) := g^m where H(m) is a one-way function. However, this protocol, as discussed in previous lecture, is secure against direct attacks but vulnerable to eavesdropping attacks.
Using Schnorr Identification Alice can convince Bob that she knows message, m, without sending the value to Bob. For this protocol, let G be a cyclic group of prime order q, with generator, g, in which the discrete log problem is assumed to be hard. *Typically a Schnorr group is used.*
Here is how the protocol works in action. Let C be a subset of Z_q.

- The key generation algorithm G runs as follows:

$$\alpha \leftarrow Z_q, \text{ u} \leftarrow g^{\alpha}$$

The verification key is vk := u, and the secret key is sk := α.

- The protocol between Alice and Bob runs as follows, where the Alice is chooses sk = α, and the verifier Bob is chooses vk = u:

  1. Alice computes $\alpha_t \leftarrow Z_q$, $u_t \leftarrow g^{\alpha_t}$, and sends u_t to Alice
  2. Bob computes c $\leftarrow$ C, and sends c to Alice
  3. Bob checks if $g^{\alpha_t} = u_t * u^c$; if so Bob outputs accept; otherwise, Bob outputs reject

If Bob accepts the message, or $g^{\alpha_t} = u_t * u^c$, then Bob accepts the conversation and this protocol satisfies the requirement for identification on the receiver.

13.2 Schnorr's Signature

Having understood how Schnorr Identification works we can use the same protocol as a signature scheme. We will make use of a secure random oracle model under the discrete log assumption in order to get random values.
Similar to Schnorr's Identification we will let G be a cyclic group of prime order q, with generator, g, in which the discrete log problem is assumed to be hard. *Typically a Schnorr group is used.* In addition we will make user of a secure hash function: M X G $\rightarrow$ C, which will behave as a random oracle where M is a message space of the signature. Here is how the signature scheme works in action.

- The key generation algorithm G runs as follows:

$$\alpha \leftarrow Z_q, \text{ u} \leftarrow g^{\alpha}$$

  The public key is pk := u, and the secret key is sk := α.

- To sign the message, m, using the secret key sk, it follows:

$$\text{S(sk, m): } = \alpha_t \leftarrow Z_q,\ u_t \leftarrow g^{\alpha_t}\ ,\ \text{c} \leftarrow \text{H(m },u_t),\ \alpha_z \leftarrow \alpha_t + \alpha\text{c}$$
$$\text{output } \sigma : = (u_t,\alpha_z).$$

- To verify a signature $\sigma = (u_t, \alpha_z)$ on a message , m, using the public key pk = u, the signature verification algorithm Bob computes c $\leftarrow$ H(m, u_t), and outputs accept if $g^{\alpha_z} = u_t{*}u^c$,and outputs reject, otherwise.

The Schnorr Signature basically computes a signature on message m of type (u_t, α_z) and the verifier Bob computer c $\leftarrow$ H(m ,u_t) where the hash function plays the role of the verifier in this protocol.
Now is this scheme secure? This scheme is secure because any forger can use two distinct signatures, (σ_1, sk_1) (σ_2, sk_2) using the same message, m and u_t, in order to solve for α_t. Lets see how this works
Given a forger F, who has access to verification key:= g^{α} and signs a message m. F generates a forgery for some (c, u_t, α_z) such that the verification passes for the given sign. We

will try to show if F manages forge a signature by learning α we contradict the discrete log assumption that given g^{α} we can't find α. For this we will create two distinct games, Game0 and Game1 such that:
Game0:

- The forger sends (m,u_t) and receives c_1 form the random Oracle model
- The forger now uses c_1 to sign message m, and send a valid signature, $\sigma_1 \rightarrow$ (c, u_t, α_{z1}).

Game1:

- The forger sends (m,u_t) and receives c_2 form the random Oracle model
- The forger now uses c_2 to sign message m, and send a valid signature, $\sigma_2 \rightarrow$ (c, u_t, α_{z2}).

Now the forger has access to two signatures of same message such that $\sigma_1 = \alpha_t + \alpha c_1$ and $\sigma_1 = \alpha_t + \alpha c_2$. Now we have two equations and two unknowns thus forger can easily compute the value of α_t which contradicts the discrete log assumption

13.3 Multi-Signature using Schnorr

In multi-signature scheme we have many signers signing the same message where each signer has a signing and a verification key, (sk, vk). If there are n signers, instead of having n different long signature we want to figure out a more compact signature which is secure. Schnorr Signature can be efficiently used for multi-signature scheme. Lets a take a look:

- The key generation algorithm G runs as follows:

  $$\text{Each signer has } \alpha \leftarrow Z_q,\ \text{u} \leftarrow g^{\alpha}$$

  Therefore for n signers there α_n signing keys and u_n verification keys
- Each signer chooses a random α_i and publishes g^{α_i} such that $u_t \leftarrow g^{\alpha_1+\alpha_2..\alpha_n}$, c $\leftarrow$ H(m,u_t,u_i, L) where L is a set of all verification keys.
- We calculate each signature sequentially such that $\alpha_z \leftarrow \alpha_{z-1}\ \alpha_{ti} + \alpha_i c_i$
- The final signature is $\alpha_z \leftarrow \alpha_t + \alpha_1 c_1 + ...\alpha_n c_n$

13.4 Commitments

In the second part of the lecture we move towards a more generalized approach to build a more elaborate type of proof systems. We discussed about **Commitments** which is basically a locked box, where Alice can put her message m inside and send the box to Bob. At some later point of time Alice sends a key to Bob which allows Bob to open the box and

read the message m. This interaction between Alice and Bob is known as a **Commitment Scheme** which consists of two algorithm: Commit and Decommit. In the commit stage Alice interacts with Bob to transfer the locked message to Bob. Similarly in the Decommit phase Alice interacts with Bob to send the key for Bob to unlock the message.
Properties of Commitment Scheme:
Hiding: The commit phase must be hiding. Let denote com(m, r) as a commit function to a message m and randomness r such that for any message, m_0 and m_1 the commit function is indistinguishable
Binding: For every string c, the probability that a PPT commiter F can decommiting a string c, with key k_0 and message m_0 and decommiting a string c, with key k_1 and message m_1 is negligible. Therefore,

$$\Pr[\ (k_1,m_1),\ (k_0,m);\ \text{decommit}(c,k_1,m_1) = 1\ )\ \text{or}\ \text{decommit}(c,k_1,m_1) = 1] \text{ is negligible}$$

In order to design a secure commitment message, m, we see to pick a random r of fixed-length and send H(r*m). We cannot send H(m) because this violates a hiding property of commitment since given an m, an adversary can easily check the hash of the message to retrieve the commitment. In order to open a commitment, we need to send r and m.

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.

References