Recitation 5: ID Schemes, Sigma Protocols, and Digital Signatures

6.5610 Spring 2023

March 10th, 2023

1 Sigma Protocols

1.1 Motivation: ID Schemes

Identification schemes are protocols for proving the identity of a given party (for example, a log in). Ideally, a client holding secret k and only that client should be able to convince the server that they hold secret k.

1.1.1 Security against eavesdropping attacks

We assume a passive, eavesdropping attacker. The attacker sees all messages on the network, and they can watch as many authentications as they wish. They then must authenticate as the client.

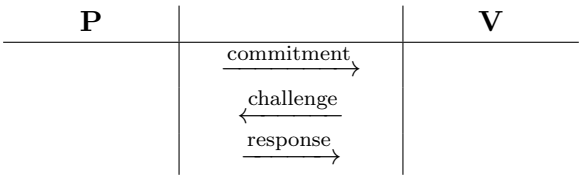
This is a weak security goal, as it assumes the attacker does not have access to the server or any server state. Breaches of this kind happen all the time in practice - a stronger security goal is necessary. Instead, we assume a passive attacker that sees all messages on the network as well as all server state. In order to achieve this goal, we use "public key" identification schemes.

1.2 Definitions

Let $\mathcal{R}$ be some relation (x, y). x is called the "witness" and y is called the "statement." If $(x, y) \in \mathcal{R}$, then the relation $\mathcal{R}$ holds for the pair (x, y).

A sigma protocol is an interactive, three message protocol one party to convince another that a pair (x, y) belongs to a relation $\mathcal{R}$ without revealing the witness x. The protocol has two algorithms, a "prover" and a "verifier." The prover knows the pair (x, y), and the verifier only knows the statement y. The prover tries to convince the verifier that it knows x such that $(x, y) \in \mathcal{R}$. The verifier algorithm outputs one bit, 1 if convinced and 0 if not.

Sigma protocols take the form the shown in the figure below. (The three arrows back and forth sort of look like Σ, giving the protocol its name!)



The notation $< P, V > (x, y)$ refers to a sigma protocol for pair (x, y) with prover algorithm P and verifier algorithm V.

Sigma protocols have the following security definitions:

- **Completeness:** For honest prover P and honest verifier V,

$$\forall (x, y) \in \mathcal{R}, \Pr[< P, V > (x, y) = 1] = 1$$

- **Knowledge soundness:** $\forall y, \forall P^*, \exists$ efficient algorithm $\mathcal{E}$ s.t.

$$\Pr[\mathcal{E}(P^*, y) \to x : (x, y) \in \mathcal{R}] \geq \Pr[< P^*, V > (; y) = 1] - negl$$

- **Honest-verifier zero knowledge (HVZK):** $\exists$ efficient algorithm $\mathcal{S}$ s.t. $\forall (x, y) \in \mathcal{R}$:

$$\{\text{transcript of } < P, V > (x, y)\} \approx_c \{\mathcal{S}(y)\}$$

1.3 Schnorr's Protocol

Assume group $\mathbb{G}$ with generator g of order q where DLOG is assumed to be hard. Schnorr's protocol is as follows[1]:

$\mathbf{P}(x, g^x)$		$\mathbf{V}(y = g^x)$
$r \leftarrow \mathbb{Z}_q$		
	$\xrightarrow{t=g^r}$	
		$c \leftarrow \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z = r + cx$		
	$\xrightarrow{z}$	
		$g^z \stackrel{?}{=} ty^c$

Schnorr's protocol is extremely useful, and with small modifications it can be used to prove many facts relating to discrete log.

1.3.1 Security Properties

- **Completeness:** Follows by construction.

$$g^z = g^{r+cx} = g^r g^{cx} = t(g^x)^c = ty^c$$

- **Knowledge soundness:** Construct an extractor as follows:
  1. Run P^* to obain transcript (t, c, z)
  2. Rewind P^* to just after the commitment t is sent
  3. Run from this point again to obtain transcript (t, c', z')
  4. Compute $\frac{z-z'}{c-c'} = \frac{(r+cx)-(r+c'x)}{c-c'} = \frac{x(c-c')}{(c-c')} = x$
- **HVZK:** Construct a simulator as follows:
  1. Draw challenge $c \leftarrow \mathbb{Z}_q$
  2. Draw response $z \leftarrow \mathbb{Z}_q$
  3. Compute $t = g^z y^{-c}$
  4. Output (t, c, z)

[1] Tip for latex-ing homework: You can use a table to make these protocol diagrams!

1.4 Fiat-Shamir Heuristic

The Fiat-Shamir heuristic is a method of turning interactive sigma protocols into non-interactive proofs by replacing the challenge with something the prover can compute themself.

Say you have a sigma protocol for witness-statement pair (x, y). Instead of asking the verifier for a random challenge in step 2, the prover instead computes $H(y, t)$ where H is a hash function.

The intuition behind the Fiat-Shamir heuristic comes from the Random Oracle Model (ROM) for hash functions. Under the ROM, the hash function is assumed to be a truly random function. Thus, the prover cannot predict what the output of $H(y, t)$ will be until computing it. Also, the output is just as random as the challenge the verifier might have drawn. Lastly, the verifier can then check that the prover actually computed the challenge correctly by checking $c \stackrel{?}{=} H(y, t)$.

1.5 Practice: Discrete Log Equivalency Proofs

Schnorr's protocol provides a way of proving that, given a generator g and a statement y, a prover knows x such that $y = g^x$. Now, we want to modify this protocol to prove that given two statements (g_1, y_1) and (g_2, y_2), a prover knows x such that $y_1 = g_1^x$ and $y_2 = g_2^x$. Modify Schnorr's protocol to meet this goal.

Tips for breaking down this problem:

1. Why doesn't running Schnorr's protocol twice in a row work here?
2. What role does the commitment serve in Schnorr's protocol? What role does the challenge and the response serve?
3. Schnorr's protocol has already handled some of the structure and algebra necessary for proving facts about discrete log in zero knowledge. How can you take advantage of that? What can you "copy and paste" to make it work for two related discrete logs?
4. How will you expresss the relation between the discrete logs as part of the protocol?

Solution:

$\mathbf{P}(x, g_1^x, g_2^x)$		$\mathbf{V}(y_1 = g_1^x, y_2 = g_2^x)$
$r \leftarrow \mathbb{Z}_q$		
	$\xrightarrow{t_1 = g_1^r, t_2 = g_2^r}$	
		$c \leftarrow \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z = r + cx$		
	$\xrightarrow{z}$	
		$g_1^z \stackrel{?}{=} t_1 y_1^c$
		$g_2^z \stackrel{?}{=} t_2 y_2^c$

1.6 Practice: Schnorr OR (also in lecture notes)

Now, modify Schnorr's protocol to prove that a prover knows one of n discrete logs, without revealing which discrete log she knows. That is, given statements $y_1 ... y_n$, prove that the prover knows x_j such that $y_j = g^{x_j}$ without revealing j.

Solution: The main idea here is to run n proofs in parallel, but use the simulator to allow the prover to simulate all but one of the proofs.

$\mathbf{P}(x_j, y_1 ... y_n)$		$\mathbf{V}(y_1 ... y_n)$
$\forall i \neq j, (t_i, c_i, z_i) \leftarrow \mathsf{Sim}(y_i)$		
$r_j \leftarrow \mathbb{Z}_q$		
$t_j = g^{r_j}$		
	$\xrightarrow{t_1,...,t_n}$	
		$c \leftarrow \mathbb{Z}_q$
	$\xleftarrow{c}$	
$c_j = c - \sum_{i \neq j} c_i$		
$z_j = r_j + c_j x_j$		
	$\xrightarrow{c_1,...,c_n}$	
	$\xrightarrow{z_1,...,z_n}$	
		$c \stackrel{?}{=} \sum_i c_i$
		$\forall i, g^{z_i} \stackrel{?}{=} t_i y^{c_i}$

2 Digital Signatures

Digital signature schemes are the public key analog of MACs. They are used to provide authenticity in the public key setting.

2.1 Definitions

A signature scheme is composed of three algorithms.

- $\mathsf{Gen}(1^n) \rightarrow (sk, pk)$
- $\mathsf{Sign}(sk, m) \rightarrow \sigma$
- $\mathsf{Ver}(pk, m, \sigma) \rightarrow \{0, 1\}$

Signature schemes have the following security properties:

- **Correctness:** $\forall (sk, pk) \leftarrow \mathsf{Gen}(1^n), \forall m \in \mathcal{M}$

$$\mathsf{Ver}(pk, m, \mathsf{Sign}(sk, m)) = 1$$

- **EUF-CMA[2] Security:** $\forall$ efficient adversaries $\mathcal{A}$, $\mathcal{A}$ wins the following game with negligible probability:
  1. Challenger runs $(sk, pk) \leftarrow \mathsf{Gen}(1^n)$
  2. Challenger sends pk to $\mathcal{A}$
  3. $\mathcal{A}$ sends challenger a message m_i
  4. Challenger signs m_i and returns σ_i
  5. Steps 3 and 4 are repeated poly-many times to the adversary's liking

[2]Existential UnForgeability under Chosen Message Attack

6. $\mathcal{A}$ outputs (m^*, σ^*)

The adversary wins if $m^* \notin \{m_i\}$ and $\mathsf{Ver}(pk, m^*, \sigma^*) = 1$.

2.2 Construction: Schnorr Signatures

Schnorr signatures are basically the Schnorr protocol made non-interactive using the Fiat-Shamir heuristic. Assume group $\mathbb{G}$ with generator g of order q. The three algorithms that make a signature scheme are constructed as follows:

- $\mathsf{Gen}(1^n)$
  1. Draw random $x \leftarrow \mathbb{Z}_q$
  2. Output x, g^x
- $\mathsf{Sign}(sk = x, m)$
  1. Draw $r \leftarrow \mathbb{Z}_q$ and compute $t = g^r$
  2. Compute $c = H(g^x, t, m)$
  3. Compute $z = r + cx$
  4. Output $\sigma = (t, c, z)$
- $\mathsf{Ver}(pk = g^x, \sigma, m)$
  1. Compute $c' = H(pk, t, m)$. Check $c' \stackrel{?}{=} c$
  2. Check $g^z \stackrel{?}{=} ty^c$; output 1 if so and 0 else.

2.3 Practice: Hash-Then-Sign

A common paradigm in digital signatures is "hash then sign." This refers to the practice of signing a hash of the message instead of signing the message itself. This problem examines the security provided by this paradigm as it interacts with signature scheme security.

1. Assume you have a signature scheme that is EUF-CMA secure. Which of the following security properties do you need from the hash function to maintain EUF-CMA security: one-wayness, collision-resistance, random oracle model.

   Solution: collision resistance. Under the EUF-CMA security game, if an adversary sees a signature σ for a message m and then is able to find a message m^* for which σ also verifies, the adversary wins the game and breaks security. If the adversary can find a collision in the hash function, the same signature will work for both of those messages. Collision resistance prevents the adversary from finding such a collision.

2. Assume the random oracle model for the hash function used in hash-then-sign. What security property does the signature scheme need in order for the hash-then-sign protocol to remain EUF-CMA secure? EUF against random message attack, EUF-CMA security, or strong EUF-CMA security.

   Solution: EUF against random message attack. Since the hash function acts as a random

oracle, any message the adversary may submit for signing in the EUF-CMA game gets converted into a random message. Therefore, the underlying signature scheme need only be secure against random message attack.