CS 355: Topics in Cryptography **Spring 2019**

Lecture 6: Sigma Protocols, Secret Sharing

Dima Kogan

1 Sigma Protocols

A more general view of Schnorr's protocol that we saw last lecture: a Sigma protocol for an NP relation $\mathcal{R}$ is an interactive protocol, in which the prover's and the verifier's inputs are x, y and y respectively, where $(x, y) \in \mathcal{R}$, and the protocol consists of three messages:

1. The prover sends the first message u, called a commitment.
2. The verifier chooses a uniformly random challenge $c \xleftarrow{R} \mathcal{C}$ from some finite challenge space $\mathcal{C}$, and sends it as the protocol's second message.
3. The prover generates a response z and sends it as the third and final message in the protocol.

Finally, we require that the verifier outputs *accept/reject* by computing some *deterministic* function on y and (u, c, z).

Prover(x, y)		**Verifier**(y)
generate commitment u		
	$\xrightarrow{\quad u \quad}$	
		generate challenge $c \xleftarrow{R} \mathcal{C}$
	$\xleftarrow{\quad c \quad}$	
generate response u		
	$\xrightarrow{\quad z \quad}$	
		output accept/reject

We require that the protocol satisfies:

1. Perfect completeness: for every $(x, y) \in \mathcal{R}$, $\Pr\left[P(x, y) \leftrightarrow V(y) = \mathsf{accept}\right] = 1$.
2. Knowledge soundness: we require the existence of an efficient extractor $\mathcal{E}$, that given two accepting transcripts (u, c, z) and (u, c', z') for y such that $c \neq c'$, outputs a witness x such that $(x, y) \in \mathcal{R}$.
3. HVZK: there exists an efficient algorithm Sim that on input (y, c)[1], where y is a statement and $c \in \mathcal{C}$ is a challenge, outputs (u, c, z) such that

$$\left\{\mathsf{Sim}(y, c) : c \xleftarrow{R} \mathcal{C}\right\} \approx_c \left\{\mathsf{View}_V(P(x, y) \leftrightarrow V(y))\right\} .$$

[1] Explicitly giving the simulator the challenge c often makes proofs easier.

In the definition above, we did not explicitly require the soundness property. The reason is that the knowledge-soundness requirement implies that when $\mathcal{C}$ is not too small, the protocol is sound, as the next lemma shows:

Lemma 1. *A Sigma protocol for an* NP *relation $\mathcal{R}$ gives an interactive proof for the language $\mathcal{L}_{\mathcal{R}}$ with soundness error at most* $1/|\mathcal{C}|$.

Proof. Completeness follows immediately. For soundness, let $y \notin \mathcal{L}_R$. We claim that for every commitment u, there exists at most one *good* challenge $c \in \mathcal{C}$ such that (u, c, z) is an accepting transcript. Otherwise, if there would have been two such challenges $c \neq c'$, running the extractor on the two transcripts (u, c, z), (u, c', z') would have resulted in a witness x for y, contradicting the fact that $y \notin \mathcal{L}_R$. Therefore, the probability that V accepts y is at most at the probability that V chooses the *good* challenge, which is at most $1/|\mathcal{C}|$. □

Advanced comment: Knowledge soundness and proofs of knowledge

Last lecture, we saw the definition of a proof of knowledge. We said that a protocol is a proof of knowledge with knowledge error ϵ, if there exists an expected-polynomial-time extractor $\mathcal{E}'$ such that for every y and every prover P^*:

$$\Pr\left[(x, y) \in \mathcal{R} : x \leftarrow \mathcal{E}'^{P^*}(y)\right] \geq \Pr\left[\langle P^*, V\rangle(y) = 1\right] - \epsilon .$$

The notation $\mathcal{E}'^{P^*}$ means that $\mathcal{E}'$ is an algorithm that gets black-box access to the algorithm P^*, including the power to rewind the prover.
It turns out that if a Sigma protocol satisfies the knowledge-soundness requirement, it is also a proof of knowledge with knowledge error at most $1/|\mathcal{C}|$. For a formal proof, see a manuscript by Damgård[a]. The general idea is that we can transform an extractor $\mathcal{E}$ that meets the knowledge-soundness requirement into a knowledge extractor $\mathcal{E}'$ as follows: $\mathcal{E}'$ runs the prover to get a commitment u, sends it a random challenge $c \xleftarrow{R} \mathcal{C}$, and obtains a response z. If (u, c, z) is not an accepting transcript, it restarts. Otherwise (when (u, c, z) is an accepting transcript), the extractor rewinds the prover to its state after it has sent the message u, sends it a fresh challenge $c' \xleftarrow{R} \mathcal{C}$, and obtains a fresh response z'. If the second transcript is also accepting and $c \neq c'$, the extractor $\mathcal{E}'$ runs $\mathcal{E}$ on (u, c, z), (u, c', z') to obtain a witness x such that $(x, y) \in \mathcal{R}$. If the second transcript is not accepting, it rewinds again, and tries another challenge. The full proof in the aforementioned manuscript requires extra care to handle small success probabilities.

[a] http://www.cs.au.dk/~ivan/Sigma.pdf

1.1 Fiat-Shamir: NIZKs in the Random Oracle Model

The Fiat-Shamir heuristic, that we've seen for Schnorr's protocol, can be applied to any Sigma protocol to obtain Non-interactive zero-knowledge proofs in the Random Oracle model.

$\mathbf{P}(x,y)$:
generate commitment u
generate challenge $c \leftarrow H(y,u)$
generate response z
output $\pi = (u,z)$

$\mathbf{V}(y,\pi=(u,z))$:
generate challenge $c \leftarrow H(y,u)$
check (u,c,z)

We say that a NIZK proof is *existentially sound* if for every **efficient** adversary $\mathcal{A}$:

$$\Pr[y \notin \mathcal{L}_{\mathcal{R}} \text{ and } V(y,\pi)=1 : (y,\pi) \leftarrow \mathcal{A}(1^{\lambda})] \leq \text{negl}(\lambda).$$

Signatures. If we bind the NIZK proof to a specific message by adding the message as an additional input to the hash function (random oracle)

$$c \xleftarrow{R} H(y,u,m),$$

we obtain a signature scheme, in which $sk=(x)$ and $pk=(y)$. We can then prove the security of the resulting signature scheme (existenital unforgeability) in the random oracle model, by showing that we can use an adversary that forges a signature to construct another adversary that breaks the identification protocol.

2 Secret Sharing

Suppose that Alice holds a secret $\alpha \in Z$, where Z is some finite set. She wants to generate a set of n shares $s_1, s_2, \ldots, s_n$, such that any t of the shares are sufficient to reconstruct the original secret α, and every subset of size $t-1$ or less reveals nothing about the secret.

Definition 2. A secret sharing scheme over Z is a pair of efficient algorithms (G, C), such that

- G is a probabilistic algorithm that is invoked as $(s_1, s_2, \ldots, s_n) \xleftarrow{R} G(n,t,\alpha)$ where $0 < t \leq n$ and $\alpha \in Z$, to generate a t-out-of-n sharing of α.
- C is a deterministic algorithm that is invoked as $\alpha \leftarrow C(s_{i_1}, \ldots, s_{i_t})$ to recover α.

We require the following two properties to hold:

- Correctness: for every $\alpha \in Z$, every set of n shares output by $G(n,t,\alpha)$, and every t-size subset $\{s_{i_1}, \ldots, s_{i_t}\}$ of the shares, we have that $C(s_{i_1}, \ldots, s_{i_t}) = \alpha$.
- Security: for every $\alpha, \alpha' \in Z$ and every subset $S \subset [n]$ of size $t-1$, the distributions $G(n,t,\alpha)[S]$ and $G(n,t,\alpha')[S]$ are identical, where we denote $G(n,t,\alpha)[S] = \left\{s_j : (s_1, \ldots, s_n) \xleftarrow{R} G(n,t,\alpha) \text{ and } j \in S\right\}$.

Note that this definition requires information-theoretic security: the two distributions for α and α' need to be identical. It could be relaxed by requiring the distributions to be computationally indistinguishable.

Example: Additive secret sharing. For $t = n$, we can construct an n-out-of-n secret sharing scheme as follows. Take $Z = \mathbb{Z}_p$. Then

- $G(n, n, \alpha)$ samples $n-1$ random shares $s_1, \ldots, s_{n-1} \xleftarrow{R} \mathbb{Z}_p$ and sets the last share as $s_n \leftarrow \alpha - \sum_{i=1}^{n-1} s_i \in \mathbb{Z}_p$.
- $C(s_1, \ldots, s_n)$ outputs $\sum_{i=1}^{n} s_i$.

Example: Combinatorial secret sharing. Let $0 < t \leq n$, and E, D be some symmetric encryption scheme. Then

- $G(n, t, \alpha)$ samples n encryption keys $k_1, \ldots, k_n$. For every $S \subseteq [n]$ of size t it creates the ciphertext $\mathsf{ct}_S = E\left(k_{i_1}, E\left(k_{i_2}, \ldots, E(k_{i_t}, \alpha) \ldots\right)\right)$ using the keys of S. Let $\mathsf{ct} = \{\mathsf{ct}_S : S \subseteq [n] \text{ s.t. } |S| = t\}$ be the collection of ciphertexts. G outputs the shares $((k_1, \mathsf{ct}), \ldots, (k_n, \mathsf{ct}))$.
- $C\left((k_{i_1}, \mathsf{ct}), \ldots, (k_{i_t}, \mathsf{ct})\right)$ decrypts $\mathsf{ct}_{\{i_1, \ldots, i_t\}} \in \mathsf{ct}$ using $k_{i_1}, \ldots, k_{i_t}$.

Note that this is now only computationally secure. The big drawback of this scheme is that the shares are exponentially large: on the order of $\binom{n}{t}$.

3 Shamir Secret Sharing

3.1 Mathematical Background

We begin by stating the following fact.

Lemma 3. *For every set of $d+1$ points $(x_0, y_0) \ldots, (x_d, y_d) \in \mathbb{F}^2$ such that $x_i \neq x_j$ for $i \neq j$, there exists a unique polynomial $f \in \mathbb{F}[x]$ of degree d such that $f(x_i) = y_i$ for every $i = 0, 1, \ldots, d$.*

Proof. Given $d+1$ points $(x_0, y_0), \ldots, (x_d, y_d)$, let

$$f(x) = a_0 + a_1 x + \ldots + a_d x^d \text{ where } a_0, \ldots, a_d \in \mathbb{F}_p$$

be a polynomial of degree d. We require:

$$\begin{aligned} f(x_0) &= a_0 + a_1 x_0 + \ldots + a_d x_0^d = y_0 \\ &\vdots \\ f(x_d) &= a_0 + a_1 x_d + \ldots + a_d x_d^d = y_d \end{aligned}$$

which we can write in matrix form as:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^d \\ 1 & x_1 & x_1^2 & \ldots & x_1^d \\ & \vdots & & & \\ 1 & x_d & x_d^2 & \ldots & x_d^d \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_d \end{bmatrix}$$

Notice that the matrix does not depend on the y-values but only on the x-values. This $(d+1) \times (d+1)$ matrix is called the Vandermonde matrix $V(x_0, \ldots, x_d)$. It's determinant is

$$\mathsf{det}(V(x_0, \ldots, x_d)) = \prod_{0 \leq i < j \leq d} (x_j - x_i),$$

which is non-zero if $x_i \neq x_j$ for every $i \neq j$. This means that this system of linear equations has a unique solution $\vec{a} = V^{-1}\vec{y}$, which gives us a unique polynomial f. In fact an explicit formula for the inverse of the Vandermonde matrix V^{-1} is known and can be used to compute the coefficient vector $\vec{a}$. $\square$

3.2 The Scheme

Shamir's t-out-of-n secret sharing scheme over $Z = \mathbb{Z}_p$, where $p > n$ is a prime, works as follows:

- $G(n, t, \alpha)$: choose random coefficients $a_1, \ldots, a_{t-1} \xleftarrow{R} \mathbb{Z}_p$ and define the polynomial:

$$f(x) = \alpha + a_1 x + \cdots + a_{t-1} x^{t-1} \in \mathbb{Z}_p[x] .$$

  Notice that f has degree at most $t-1$ and that $f(0) = s$. For $i = 1 \in [n]$ compute $y_i \leftarrow f(i) \in \mathbb{Z}_q$, and define $s_i = (i, y_i)$. Output the n shares $s_1, \ldots, s_n \in \mathbb{Z}_p^2$.

- $C(s_{i_1}, \ldots, s_{i_t})$: these t distinct points on the polynomial f completely determine f. The algorithm interpolates the polynomial f and outputs $\alpha \leftarrow f(0)$ (which is also the constant term of the polynomial).

To prove security, let α be the message. We show that the values of the y-coordinates of the shares are distributed uniformly and independently (both of each other and of α) over $\mathbb{Z}_p$. To this end, consider the map that sends a choice of coefficients $(a_1, \ldots, a_{t-1}) \in \mathbb{Z}_p^{t-1}$ to the y-coordinates of the shares $(y_{i_1}, \ldots, y_{i_t-1}) \in \mathbb{Z}_p^{t-1}$. This map is one-to-one, since the t points $(0, \alpha), (i_1, t_{i_1}), \ldots, (i_{t-1}, t_{i_{t-1}})$ uniquely determines a polynomial. Therefore, if we choose the coefficients independently uniformly at random, the $(t-1)$ shares are also distributed independently uniformly at random, and in particular are independent of the message α.

3.3 Application: Threshold Decryption

In any public-key encryption scheme, one can use t-out-of-n Shamir secret sharing to share the secret decryption key between n servers. Then, anyone can encrypt a message to the servers using the public key, but it takes a coalition of t servers to decrypt a ciphertext: t servers recombine the secret key and decrypt.

This creates a single point of failure when recombining the secret: an adversary that compromises the combiner sees the secret key in the clear. A threshold decryption scheme allows a coalition of t out of n servers to decrypt any ciphertext but without having the secret at a single location at any point in time.

We'll see an construct a threshold decryption scheme for the ElGamal encryption scheme.

Reminder: multiplicative ElGamal encryption scheme. The secret key is $\alpha \xleftarrow{R} \mathbb{Z}_q$ and the public key is $u \leftarrow g^\alpha \in \mathbb{G}$.

- $E(u, m \in \mathbb{G})$: choose $\beta \xleftarrow{R} \mathbb{Z}_q$, set $v \leftarrow g^\beta$, $w \leftarrow u^\beta$, $e \leftarrow w \cdot m$, Output (v, e).

- $D(\alpha, (v, e))$: compute $w \leftarrow v^\alpha$ and output e/w.

To turn this to a threshold decryption scheme, we use Shamir secret sharing to share the secret key α and get n shares $(1, y_1), \ldots, (n, y_n)$. Give each of the n servers its share.

To decrypt a ciphertext (u, e), the servers need to compute $w \leftarrow v^{\alpha}$, but they want to do this without reconstructing α at any single point. Recall that the coefficeints of the polynomial can be computed as:

$$\begin{bmatrix} \alpha \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix} = V^{-1} \cdot \begin{bmatrix} y_{i_1} \\ y_{i_2} \\ \vdots \\ y_{i_t} \end{bmatrix},$$

where V^{-1} is the inverse Vandermonde matrix, and $(x_{i_1}, y_{i_1}), \ldots, (x_{i_t}, y_{i_t})$ are t distinct points on the polynomial. In particular

$$\alpha = \sum_{j=1}^{t} b_{1j} y_{i_j},$$

where $b_{11}, \ldots, b_{1t}$ are the elements of the first row of the matrix V^{-1}. Therefore,

$$w = v^{\alpha} = v^{\sum_{j=1}^{t} b_{1j} y_{i_j}} = \prod_{j=1}^{t} v^{b_{1j} y_{i_j}} = \prod_{j=1}^{t} \left(v^{y_{i_j}}\right)^{b_{1j}}.$$

This suggests a method for threshold decryption of a ciphertext (v, e):

- Server i_j for $j \in [t]$ uses its share y_j to compute $w_j \leftarrow v^{y_{i_j}}$ and sends (i_j, w_j) to the recombiner.
- The recombiner gets t partial decryptions $w_1, \ldots, w_t$ from servers $i_1, \ldots, i_t$. It computes the first row $\vec{b}_1$ of the inverse Vandermonde matrix $V^{-1}(i_1, \ldots, i_t)$, computes $w = \prod_{j=1}^{t} w_j^{b_{1j}}$, and decrypts the ciphertext as $m \leftarrow e / w$.

For a full security proof, see Boneh-Shoup, Chapter 11.6.2.