**Stanford University**
**Topics in Cryptography (CS 355)** **Lecture #3**
**Instructor: Dr. Lior Rotem** **April 11, 2023**

**Disclaimer** *These lecture notes are aimed to serve as a supplementary resource, and are not written a replacement for attending the in-person lecture. Some material might appear here in a more sketched form than in the lecture, and vice versa. These notes probably contain typos and might even contain errors. If you find any, please let me know.*

**Outline** *In this lecture, we explore a very useful and fundamental cryptographic primitive: commitment schemes. We present an example of how such schemes can be used in the context of collective randomness generation protocols. We then present several constructions, including one from (almost) minimal complexity assumptions and two very efficient and useful constructions from specific assumptions. Along the way, we define and discuss the random oracle model (ROM).*

## 1 Commitment Schemes and Coin Flipping

**A motivating example: A coin flipping protocol.** Alice and Bob are two CS grad students at Stanford. They agreed to have lunch together, but there's a problem: Alice wants to go to Bytes, but Bob prefers Blend. The only conceivable solution to this conundrum is to flip a coin. However, Alice's office is on the first floor of Gates, and Bob's is on the fourth. Meeting in person seems beyond reach. But what can they do? If Alice flips the coin and reports the result to Bob on Slack, can Bob really trust her? If Bob flips the coin and reports the result to Alice, can Alice trust the result? Being students in CS 355, they agree to run a cryptographic protocol. First, they need to learn about commitment schemes.

**Commitment schemes.** Intuitively, a commitment scheme is the digital analog of a locked opaque box. Alice can lock some value, say a bit $b$, inside this box and send it to Bob. At this point, Bob knows nothing about the value inside the box. This property is called *hiding*. In due time, Alice can send the key to Bob, allowing him to open the box and learn $b$. Note that once the box is sent, the only value that Bob can find inside it is $b$. This is true regardless of the key that Alice provides him with (at worst, Alice can send a wrong key that does not open the box at all). This property is called *binding*. Let us look at a more formal definition.

A non-interactive commitment scheme is a pair $\Pi = (Setup, Commit)$ of randomized algorithms:

- $Setup(1^\lambda) \to pp$: takes as input the security parameter $\lambda \in \mathbb{N}$ (in unary) and outputs public parameters $pp$ (examples for $pp$ that we will encounter include a random bit-string or a description of a cryptographic group).
- $Commit(pp, m; r) \to c$: takes as input $pp$, a message $m$ from the message space $\mathcal{M}(pp)$ (that is, the message space may depend on $pp$), and randomness $r$. The $Commit$ algorithm outputs a commitment $c$.

We do not specify an explicit verification algorithm, since given $pp, c, m$ and $r$, one can verify that $c$ is indeed a commitment to $m$ by asserting that $c = Commit(pp, m; r)$. Note that this verification procedure guarantees perfect correctness: If the sender reports the true randomness used to generate $c$, then the receiver will always accept.

**Security:** Commitment schemes should satisfy two security properties:

1. **Hiding**: The commitment $c$ should hide the message $m$. Formally, for every PPT adversary $A$ there exists a negligible function $\nu(\cdot)$ such that

$$\Pr\left[ b' = b \ : \begin{array}{c} pp \xleftarrow{\$} Setup(1^\lambda) \\ (st, m_0, m_1) \xleftarrow{\$} A(pp) \\ b \xleftarrow{\$} \{0,1\} \\ c \xleftarrow{\$} (pp, m_b) \\ b' \xleftarrow{\$} A(st, c) \end{array} \right] \leq \frac{1}{2} + \nu(\lambda)$$

   for all sufficiently large $\lambda \in \mathbb{N}$.

2. **Binding:** Given $pp$, it should be infeasible to produce a commitment $c$ with two different openings $m_0, r_0$ and $m_1, r_1$ (for $m_0 \neq m_1$). for every PPT adversary $A$ there exists a negligible function $\nu(\cdot)$ such that

$$\Pr\left[ \begin{array}{c} \forall b \in \{0,1\} \ Commit(pp, m_b; r_b) = 1 \\ and \\ m_0 \neq m_1 \end{array} \ : \ \begin{array}{c} pp \xleftarrow{\$} Setup(1^\lambda) \\ (c, m_0, m_1, r_0, r_1) \xleftarrow{\$} A(pp) \end{array} \right] \leq \nu(\lambda)$$

   for all sufficiently large $\lambda \in \mathbb{N}$.

Three remarks regarding the above definition are in order:

1. *Statistical security*: Either hiding or binding can hold statistically, that is, against unbounded adversaries. However, you cannot have both statistical hiding and statistical binding (make sure that you understand why).

2. *Verification and correctness*: It is possible to generalize the definition to include an arbitrary (typically deterministic) verification algorithm $Verify$. Then, $Commit$ also outputs some decommitment string $d$, and $Verify$ then takes in $d$ as input instead of $r$. In this case, correctness needs to be required explicitly: Informally, $Verify$ should always accept $pp, m, c, d$ that were honestly generated.

3. *Interactive commitments*: The definition can be generalized by replacing the $Commit$ algorithm with an interactive protocol between the sender and the receiver of the commitment. The commitment $c$ is then the transcript of this protocol.

**Back to coin flipping.** Equipped with the notion of a commitment scheme, we can now present the coin flipping protocol. The protocol uses a commitment scheme $\Pi = (Setup, Commit)$. Before the protocol is executed, the $Setup$ algorithm is used to produce public parameters $pp$. The protocol then proceeds in three rounds:

1. Alice samples a random bit $b \xleftarrow{\$} \{0,1\}$ and computes a commitment $c$ to it using randomness $r$. That is, $c \leftarrow Commit(pp, b; r)$. It then sends $c$ to Bob.
2. Bob samples a bit $b' \xleftarrow{\$} \{0,1\}$ and sends it to Alice.
3. Alice sends $b$ and $r$ to Bob. Bob verifies that $Commit(pp, b; r) = c$ and if not, he aborts.
4. Output: If not aborted, the parties output the bit $b \oplus b'$.

The security guarantee of this protocol is that if one of the parties is honest, then the output bit is essentially unbiased. We briefly and informally sketch a proof for this claim. First, we assume that Alice is honest and argue that Bob cannot bias the output bit. By the hiding property, the commitment $c$ reveals nothing about the bit $b$. Hence, when Bob chooses $b'$ it knows nothing about $b$, meaning $b'$ is chosen independently of $b$. Hence, since $b'$ and $b$ are independent, and $b$ is uniformly random (Alice is honest), the output bit $b \oplus b'$ is also uniformly random. Now assume that Bob is honest. We argue that Alice cannot bias the output bit. By the binding property, Alice can open $c$ to (at most) a single bit $b$. Informally, this means that $c$ determines $b$, which is chosen before $b'$ and is hence independent of $b'$. Since $b'$ is uniformly random (Bob is honest) this means that the output bit is as well.

It should be emphasized that the above sketch is very informal and the actual proof. The actual proof more elaborate and is by reductions to the hiding and binding properties of $\Pi$. You are encouraged to try and prove it yourself, or take a look at the original paper.

## 2 A Simple Commitment from Injective OWFs

The first construction of a commitment scheme that we will see is based on the existence of an injective one-way function. Let $f$ be such a function and let $\mathsf{hcb}$ be the associated hardcore predicate (recall lecture 1). Consider the following commitment Scheme $\Pi = (Setup, Commit)$. The *Setup* algorithm does nothing and just returns the security parameter $1^\lambda$. In order to commit to a bit $b \in \{0,1\}$, the *Commit* algorithm samples an input $x$ from the domain of $f$. It then outputs the commitment $c = (f(x), \mathsf{hcb}(x) \oplus b)$.

We argue that this is a hiding and biding commitment scheme:

- **Binding:** The scheme is statistically binding (it is actually *perfectly* binding, meaning that for any potentially unbounded adversary, the negligible function $\nu$ from the binding definition is simply 0). Observe that the randomness used by *Commit* is the sampled $f$-input $x$. Since $f$ is injective, for any commitment $c = (c_1, c_2)$, there is at most one value $x$ such that $f(x) = c_1$. This value $x$ and $c_2$ determine the committed bit $b$.
- **Hiding:** We argue that the second coordinate of $c = (c_1, c_2)$ is pseudorandom and hence reveals no information of the committed bit $b$. By definition of a hardcore predicate, $\mathsf{hcb}(x)$ is pseudorandom even given $f(x)$. Hence $c_2 = \mathsf{hcb}(x) \oplus b$ is also pseudorandom given $c_1 = f(x)$. In other words, $c$ is computationally indistinguishable from a pair $(f(x), b')$ where $b'$ is a uniformly random bit, independent of both $x$ and $b$.

## 3 A Hash-Based Commitment and the ROM

A heuristic approach for constructing commitment schemes that is often used in practice is to rely on a cryptographic hash function $\mathsf{H}$ (for example, SHA-3). As before the *Setup* algorithm does nothing and returns $1^\lambda$.[1] To commit to a message $m$, the *Commit* algorithm samples a random bit-string $r \xleftarrow{\$} \{0,1\}^\lambda$ (where $\lambda$ is the security parameter) and computes the commitment $c \leftarrow \mathsf{H}(m, r)$. We will assume that the output length of $\mathsf{H}$ on such an input is at least $\lambda$-bit long.

The hope is that the hash function $\mathsf{H}$ is complicated enough, so that: (1) $c$ essentially reveals nothing about $m$, making the commitment scheme hiding, and (2) it is infeasible to find collisions in $\mathsf{H}$, making the commitment scheme binding. But how can we prove that?

**The random oracle model.** The random oracle (ROM for short) is an idealized model that allows us to formally reason about constructions that use cryptographic hash functions. Over the past 30 years, this model has become extremely popular for analyzing the security of hash-based cryptographic constructions.

The idea is to treat $\mathsf{H}$ as if it was a truly random function. This is captured by allowing all algorithms, including the adversary, *oracle access* to an oracle $\mathcal{O}$. If $\mathsf{H}$ maps inputs from a domain $\mathcal{D}$ to a co-domain $\mathcal{R}$, then $\mathcal{O}$ is a function chosen uniformly at random from the set of all functions from $\mathcal{D}$ to $\mathcal{R}$. By oracle access, we mean that algorithms can send inputs $x \in \mathcal{D}$ to $\mathcal{O}$ and receive $\mathcal{O}(x)$ in response.

*Defining security in the ROM.* A security definition in the standard model (i.e., the model without a random oracle) naturally translates into a security definition in the ROM. This is typically done through two modifications:

1. The algorithms of the construction (in our case, of the commitment scheme) query the oracle $\mathcal{O}$ instead of computing $\mathsf{H}$.
2. The adversary is given oracle access to the oracle $\mathcal{O}$.
3. The probability that the adversary wins the security game is also taken over the choice of $\mathcal{O}$.

**Proving security in the ROM.** We now turn to prove the hiding and binding of the above hash-based commitment scheme. In both cases, we bound the advantage of the adversary as a function of the number of oracle queries it performs. Hence, we actually prove unconditional security *in the ROM*. As we will see later in the course, this will not always be the case in proofs within the ROM. It is important to remember however, that instantiating the random oracle with a concrete hash function requires that we heuristically assume that this hash function is "as good" as a random function for our needs, which is a computational assumption.

*Hiding:* Let $A$ be an adversary that makes $q = q(\lambda)$ queries to the random oracle $\mathcal{O}$. $A$ outputs two messages $M_0$ and $M_1$ and a state $ST$ (we use upper case letters to emphasize the fact that these are random variables). Then, a bit $B$ is chosen, randomness $R$ is chosen from $\{0,1\}^\lambda$ and a commitment $C \leftarrow \mathcal{O}(M_B, R)$ is computed and returned to $A$. Finally, $A$ outputs a bit $B'$.

[1]In practice, it is advisable to have the *Setup* algorithm choose a *salt* string $s$ to be appended to all inputs to $\mathsf{H}$. This helps in preventing preprocessing attacks. For simplicity, we ignore this fact here.

Let $\mathsf{HIT}$ denote the event in which $A$ queries $\mathcal{O}$ with an input that ends with $R$. Observe that conditioned on the complementing event $\overline{\mathsf{HIT}}$, the view of $A$ is statistically independent of the bit $B$. That is, conditioned on the view of $A$ (which includes the security parameter $\lambda$, $A$'s randomness, and the commitment $C$), the probability that $B = 0$ is equal to the probability that $B = 1$. Hence $\Pr\left[B = B' \middle| \overline{\mathsf{HIT}}\right] = 1/2$.

So we are left with bounding $\Pr[\mathsf{HIT}]$. For $i \in \{1, \dots, q\}$, let $R_i$ denote the last $\lambda$ bits of $A$'s $i$th query to $\mathcal{O}$, and let $\mathcal{R}_i = \{R_1, \dots, R_i\}$ (note that $\mathcal{R}_0 =$). Then $\mathsf{HIT}$ is the event in which $R \in \mathcal{R}_q$. Let $i \in \{1, \dots, q-1\}$ and assume $R \notin \mathcal{R}_i$. In this case, according to the view of $A$, $R$ is uniformly-distributed in a superset of $\{0,1\}^\lambda \setminus \mathcal{R}_i$. Hence

$$Pr\left[R \in \mathcal{R}_{i+1} \mid R \notin \mathcal{R}_i\right] = Pr\left[R_{i+1} = R \mid R \notin \mathcal{R}_i\right] \le \frac{1}{2^\lambda - i}.$$

By a union bound, we obtain that

$$\begin{aligned} Pr\left[\mathsf{HIT}\right] &= Pr\left[R \in \mathcal{R}_q\right] \\ &\le \sum_{i=1}^{q} Pr\left[R_{i+1} = R \mid R \notin \mathcal{R}_i\right] \\ &\le \frac{q}{2^\lambda - q}. \end{aligned}$$

Putting everything together, we have that $\Pr[B' = B] \le \frac{1}{2} + \frac{q}{2^\lambda - q}$, which is negligible for any polynomial $q$.

*Binding:* As before, let $A$ be an adversary that makes $q = q(\lambda)$ queries to $\mathcal{O}$. $A$ outputs $C$ together with $M_0, R_0$ and $M_1, R_1$ and wins if $\mathcal{O}(M_0, R_0) = C = \mathcal{O}(M_1, R_1)$ and $M_0 \neq M_1$. In other words, $A$ winning means that it has found a collision in $\mathcal{O}$. So we wish to bound the probability for this event. Recall that the output length of $\mathcal{O}$ is at least $\lambda$ bits long; assume that it is exactly $\lambda$ bits long. Hence, if $A$ outputs without querying $\mathcal{O}$ with $(M_0, R_0)$ and with $(M_1, R_1)$, the probability that $A$ wins is $2^{-\lambda}$. Since $A$ makes at most $q$ distinct queries to $\mathcal{O}$, and each response is uniformly-distributed in $\{0,1\}^\lambda$ independently of the other responses of $\mathcal{O}$, the birthday bound bounds the probability that two of these responses collide. Concretely, this probability is at most $q^2 \cdot 2^{-\lambda}$. Hence, the overall probability that $A$ breaks binding is at most $(q^2 + 1) \cdot 2^{-\lambda}$, which is negligible in $\lambda$ for any polynomial $q$.

**How to think about security proofs in the ROM?** Proving security within the ROM, and then instantiating the random oracle with a concrete hash function is a heuristic. there is no formal justification for assuming that a security proof in the ROM implies security in the real world (when instantiating the scheme with a concrete hash function). For example, in the above proofs we assumed that as long as $A$ hadn't queried $\mathcal{O}$ on some input $x$, then the value $\mathcal{O}(x)$ is uniformly random in the view of $A$. It is unclear what this even means in the real world, when $\mathcal{O}$ is replaced with some concrete hash $\mathsf{H}$. This is actually a very mild use of the ROM, and more advanced proof techniques use the ROM in even ways which are even more unrealistic. We will see examples later in the course. Worse still, there are (contrived) examples for cryptographic constructions that are secure in the ROM, but are *provably insecure* in the real world when $\mathcal{O}$ is instantiated with a concrete hash function, *for any choice* of hash function.

So why use the random oracle heuristic? Firstly, it allows for more efficient cryptographic schemes which are provably secure (albeit in the ROM). Though these scheme can be defined and assumed secure without the random oracle heuristic, the convention is that it is better to have a security proof in this idealized model than no proof at all. It is not obvious that this is the case, and there are other idealized models in which security proofs are considered more "fishy". Still, the random oracle heuristic is considered reasonable since it provides a security proof "up to the choice of hash function". That is, breaking a scheme $\Pi$ proven secure in the ROM requires that the breaker understands something non-trivial the hash function $\mathsf{H}$ used to instantiate the random oracle. Moreover, in such cases, one can just swap out $\mathsf{H}$ and use a different, arguably more secure, hash function. To this day, there have been no attacks on natural (non-contrived) schemes proven secure in the ROM.

## 4 Pedersen Commitments in Discrete-Log Hard Groups

We now present a commitment scheme, due to Pedersen, in cyclic groups in which the discrete log problem is assumed to be hard. We will see examples for such groups later in the course, but for now it will be convenient to think about them in an abstract manner. We will just assume that we have some cyclic group $\mathbb{G}$ of order $p$, generated by $g \in \mathbb{G}$. We assume that the discrete log problem is hard in $\mathbb{G}$. This means that given $(\mathbb{G}, p, g, g^x)$ for a uniformly random $x \overset{\$}{\leftarrow} \mathbb{Z}_p$ it should be infeasible to compute $x$.

**Pedersen commitments.** The *Setup* algorithm samples a group element $h \overset{\$}{\leftarrow} \mathbb{G}$ and outputs the public parameters $pp = (\mathbb{G}, p, g, h)$. The commitment allows to commit to elements in $\mathbb{Z}_p$. To commit to an element $x \in \mathbb{Z}_p$, the *Commit* algorithm samples a random $r \overset{\$}{\leftarrow} \mathbb{Z}_p$ and outputs $c \leftarrow g^r \cdot h^x$.

*Hiding:* Pedersen commitments are perfectly hiding. The commitment $c$ outputted by *Commit* is just a uniformly random group element in $\mathbb{G}$. In particular, it is statistically independent of the value $x$ to which $c$ is a commitment.

*Binding:* The binding property of Pedersen commitments is proven by reduction to the discrete log problem: An algorithm $A$ which breaks binding can be converted into an algorithm $B$ that computes the discrete log of $h$ with respect to $g$; that is, it computes the unique $\mathbb{Z}_p$ element $x$ such that $h = g^x$. The algorithm $B$ gets as input $(\mathbb{G}, p, g, h = g^x)$ and is defined as follows:

1. Set $pp \leftarrow (\mathbb{G}, p, g, h)$ and invoke $A(pp)$.
2. When $A$ outputs $c, x_0, r_0, x_1, r_1$ check that $g^{r_0} \cdot h^{x_0} = c = g^{r_1} \cdot h^{x_1}$ and $x_0 \neq x_1$. If the either condition does not hold, abort.
3. If not aborted, we know that $g^{r_0 - r_1} = h^{x_1 - x_0}$. In other words, $x = (r_0 - r_1) \cdot (x_1 - x_0)^{-1}$ is the discrete log of $h$ relative to $g$. So $B$ outputs $x$.

Note that whenever $A$ breaks binding, $B$ breaks the discrete log problem. In particular, if $A$ has a non-negligible probability of breaking binding, $B$ has a non-negligible probability of breaking discrete log.