Privacy Enhancing Technologies FS2025
Lecture 2-3-4 – Commitment Schemes

Florian Tramèr

AGENDA

1. Rock-paper-scissors over the Phone
2. Commitment Schemes
3. Random Oracles
4. Pedersen Commitments
5. Pseudorandom Functions

1 Rock-Paper-Scissors Over the Phone

Today we'll see a very simple form of privacy primitive, which will play a key role in more advanced crypto schemes we'll cover later. It is also a nice illustration that privacy techniques can have applications in protocol design beyond "protecting personal data".[1]

Suppose that you and your friend are on the phone and you need to choose where to have dinner. You want Chinese and they want Italian, and so you need a tie breaker. You decide to play a game of rock-paper-scissors.[2]

If you were next to each other, you could just play a round of rock-paper-scissors and make sure there is no cheating. But how would we do this over the phone? Whenever you start announcing "rock..." your friend might quickly change their mind, announce "paper", and claim there was a slight delay on the line.

So we need a way to "synchronize" the two announcements, so that one party cannot choose their answer based on the other party's answer. How could we do this? Let's consider two natural cryptographic solutions that don't quite work:

- *Hashing:* You pick your choice $x \leftarrow \{0,1,2\}$ and send a cryptographic hash $h \leftarrow H(x)$ to your friend. Your friend picks their own choice $x' \leftarrow \{0,1,2\}$ and sends it to you in the clear. You then reveal x and your friend checks that $h = H(x)$.

  Unfortunately, you won't get your Chinese food! Your friend can just check if $h = H(0)$, $h = H(1)$, or $h = H(2)$ and then set x' accordingly to win. The problem is that the hash doesn't *hide* your choice x.

- *Encryption:* So let's use a stronger form of data hiding, encryption. You pick a key k for a semantically-secure encryption scheme and you send $c = \texttt{Enc}(k, x)$ to your friend, who sends you x' as before. You then reveal (k, x) and your friend checks that $\texttt{Dec}(k, c) = x$.

[1] We'll see more examples of this when we talk about Differential Privacy in the second half of the course.

[2] The original application proposed by Blum [Blu83] was coin flipping over the phone. But an actual fair coin flip is a tiny bit more complicated in practice, since you need to guard against biased coins. So the actual protocol ends up essentially being a form of rock-paper-scissors.

The attack above no longer works. Because of the encryption scheme's randomness, your friend cannot tell if you encrypted a 0, 1, or 2. The problem is that now *you* can cheat! Specifically, you might be able to find a key k' such that $\mathtt{Dec}(k', c) \neq x$. For example, suppose you used the most secure encryption scheme of all, the one-time pad. So you sent $c = x + k \pmod 3$ to your friend, where k is a random value in $\{0, 1, 2\}$. But then, after seeing x', you can decide to send any other value $(x + \Delta, k - \Delta)$ to your friend. Chinese always wins! The issue here is that most encryption schemes are not *binding*: you can selectively choose a key to decrypt to any message.

To resolve this issue, we will need a primitive that is both *hiding* and *binding*. This is called a *commitment scheme* [Blu83]. You can think of a commitment scheme as a box that you can lock with a key. Once you lock your answer in the box, you can give it to your friend, and thereafter you cannot change your answer and your friend cannot learn anything about it. Later, you can give your friend the key to reveal your answer.

2 Commitment Schemes

Before we define commitment schemes, we introduce the notion of *indistinguishability*.

Indistinguishability. Very often in a PETS, the adversary observes the output of a randomized process applied to data, i.e., some random variable coming from some distribution D_0 (we'll sometimes call this the *adversary's view*). To argue privacy, we show that the distribution of the adversary's view is hard to distinguish from a different distribution D_1, which is typically very easy to reason about (e.g., it's a distribution that doesn't depend on the private data at all, and so privacy is immediate). This notion of indistinguishability can be either *statistical* or *computational*:

Definition 1 (Indistinguishability)**.** Let D_0 and D_1 be two distributions (formally, these distributions are parameterized by a security parameter λ).

We say that D_0 and D_1 are *statistically indistinguishable*, denoted $D_0 \equiv D_1$, if the distributions are identical (or negligibly close). This means that an *unbounded* adversary (with access to a polynomial number of samples) cannot distinguish the two distributions with non-negligible probability (formally, this is equivalent to saying that the total-variation distance between the two distributions is negligible in λ).

We say that D_0 and D_1 are *computationally indistinguishable*, denoted $D_0 \overset{c}{\approx} D_1$, if no PPT (probabilistic polynomial-time) adversary can tell the difference between the two distributions with non-negligible probability. More precisely, we define two events W_0 and W_1 of the adversary guessing 1 for the two possible "worlds":

$$W_0 := \left\{\mathcal{A}(x) = 1 : x \leftarrow D_0\right\} \quad \text{and} \quad W_1 := \left\{\mathcal{A}(x) = 1 : x \leftarrow D_1\right\}$$

Then, we define the *advantage* of the adversary as $\mathrm{Adv}[\mathcal{A}] := |\Pr[W_0] - \Pr[W_1]|$. The distributions are computationally indistinguishable if $\mathrm{Adv}[\mathcal{A}] = \mathsf{negl}(\lambda)$.

A formal definition of commitment schemes.

Definition 2 (Commitment scheme)**.** A commitment scheme is an algorithm $\texttt{Commit} : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$ that takes in a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$ and outputs a commitment $c \in \mathcal{C}$ such that $\texttt{Commit}(m, r) = c$.

Such a scheme should satisfy two properties:

(Statistical) Hiding: the commitment c should leak nothing about the message m:

$$\forall m_0, m_1 \in \mathcal{M}, \quad \{\texttt{Commit}(m_0, r) : r \leftarrow^{\$} \mathcal{R}\} \equiv \{\texttt{Commit}(m_1, r) : r \leftarrow^{\$} \mathcal{R}\}$$

(Computational) Binding: After sending the commitment c, the adversary cannot "change their mind." No PPT adversary $\mathcal{A}$ can produce $m_0, m_1 \in \mathcal{M}$ and $r_0, r_1 \in \mathcal{R}$ such that

$$\texttt{Commit}(m_0, r_0) = \texttt{Commit}(m_1, r_1) \text{ and } m_0 \neq m_1.$$

We can also define commitment schemes where the hiding property is computational and the binding property is statistical (i.e., even an unbounded adversary cannot break the binding property). But we cannot get both properties to be statistically secure at the same time (convince yourself of this!)

A note on terminology: while we define commitment schemes here as having a single algorithm, `Commit`, in practice commitment schemes are used as a two-stage protocol:

1. One party sends a *commitment* c to a message m to the other party.
2. The same party later sends a *opening* r to the commitment. The other party then checks that $\texttt{Commit}(m, r) = c$.

2.1 The Simplest Commitment Scheme in the World

Remember our failed attempt at coin flipping over the phone, where you sent a hash $H(b)$ to your friend? Well, here's what you should have done:

$$\texttt{Commit}(m, r) = H(m, r).$$

That's it! So simple...

How do we prove this is secure? Unfortunately, "standard" security properties of hash functions (e.g., collision resistance) are not enough.[3] To prove security, we will need

The Random Oracle Model

2.2 The Random Oracle Model

Many (most) cryptographic schemes that are used in practice rely on a heuristic: these schemes make use of a hash function (e.g., like SHA-3) which is very complicated to analyze. Therefore, when proving security, we pretend that this hash function is actually a

[3] To provide some intuition, consider a dummy cryptographic hash function that always appends (part of) the message m to the end of the hash. This still satisfies all the properties we want from a hash function (e.g., one-wayness, collision resistance), but it is definitely not hiding.

completely random function $H : \mathcal{X} \to \mathcal{Y}$, where for each $x \in \mathcal{X}$, the output $H(x)$ is sampled uniformly at random from $\mathcal{Y}$ [BR93].

Of course, this is not true in practice. A truly random function would require exponential space to define. But this trick is super useful to prove security. Let's see an example for our previous commitment scheme:

Theorem 1. The commitment scheme $\texttt{Commit} : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ defined as $\texttt{Commit}(m, r) := H(m, r)$ is computationally hiding and computationally binding in the random oracle model (where $|\mathcal{K}|$ and $|\mathcal{C}|$ are of size $2^{\text{poly}(\lambda)}$).

Proof.

Computational Hiding: If H is a random function, then $H(m, r)$ is uniformly random over $\mathcal{C}$ as long as the adversary doesn't evaluate H on (m, r). If the adversary only makes a polynomial number of queries, this happens with negligible probability (i.e., on the order of $Q/2^{\text{poly}(\lambda)}$ for Q random oracle queries).

Computational Binding: To break binding, the adversary has to find $m, m' \in \mathcal{M}$ and $r, r' \in \mathcal{K}$ such that $H(m, r) = H(m', r')$. However, the outputs of a random function are uniformly distributed in the output space $\mathcal{C}$, so this happens with negligible probability if the adversary only makes a polynomial number of queries. □

Controversy: The Random Oracle Model is somewhat controversial. The reason is that we know (for sure) that any hash function we'll use in practice is most definitely *not* a random function. This is also why we call this a model and not an assumption (such as "Factoring is hard"). The latter is possibly true (and indeed that's what we assume), while we know for certain that the first is false (but it's a convenient model to work in).

We can build very simple and efficient schemes that are secure in this model. The heuristic is then to replace the random function by a *suitable hash function* when deploying the scheme. To break the scheme, the adversary has to exploit some property of the hash function that distinguishes it from a random function. Since we don't know of such properties for cryptographic hash functions such as SHA-3, we believe the scheme remains secure.

Some (more theoretically inclined) cryptographers are not quite happy with this heuristic, and prefer to build schemes that can be proven secure under specific hardness assumptions. Unfortunately, these schemes are then often (much) more complicated and harder to deploy. It is also known that in some (highly contrived) cases, schemes that are proven secure in the random oracle model are actually *insecure* when instantiated with *any* concrete hash function [CGH04].

2.3 Pedersen Commitments

Let's now see a commitment scheme where we don't need a random oracle model. This scheme [Ped91] will be less efficient than the previous one, although still quite practical.

Definition 3 (Pedersen commitment)**.**

Setup: Let $\mathbb{G}$ be a group of prime order q, and let $g, h \in \mathbb{G}$ be generators *such that the discrete log between g and h (i.e., $g^x = h$) is unknown.*

Commitment: $\texttt{Commit} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ is defined as

$$\texttt{Commit}(m, r) := g^m h^r.$$

Why is this scheme cool? It is *linearly homomorphic*, i.e., you can combine commitments to different messages and get a commitment to the sum:

$$\texttt{Commit}(m_0, r_0) \cdot \texttt{Commit}(m_1, r_1) = \texttt{Commit}(m_0 + m_1, r_0 + r_1).$$

We can thus *compute* (linear functions) over hidden data!

To define the security of this scheme, we need to define a new problem:

Definition 4 (Discrete Logarithm Problem (DLP))**.** Let $\mathbb{G}$ be a group of prime order q, with generator g. Given an element $g^x \in \mathbb{G}$, where x is uniformly random in $\mathbb{Z}_q$, the DLP problem is to find x.

We define the advantage of an adversary $\mathcal{A}$ as

$$\text{AdvDLP}(\mathcal{A}, \mathbb{G}) = \Pr[\mathcal{A}(g^x) = x : x \xleftarrow{\$} \mathbb{Z}_q].$$

The Discrete Logarithm Assumption in $\mathbb{G}$ says that $\text{AdvDLP}(\mathcal{A}, \mathbb{G}) = \mathsf{negl}(\lambda)$ for all PPT adversaries $\mathcal{A}$.

Theorem 2. The Pedersen commitment scheme is statistically hiding. It is also computationally binding assuming the DL assumption in $\mathbb{G}$. In particular, for every adversary $\mathcal{A}$ that breaks the binding property of the Pedersen commitment scheme, there exists an adversary $\mathcal{B}$ that solves the DLP such that

$$\text{AdvDLP}(\mathcal{B}, \mathbb{G}) = \text{AdvBind}(\mathcal{A}, \mathbb{G})$$

Proof.

Hiding: The scheme is perfectly hiding. We have:

$$\texttt{Commit}(m, r) = g^m h^r = g^m \cdot g^{x \cdot r} = g^{m + x \cdot r}.$$

Given any commitment $c = g^\alpha$, and any message m, there exists exactly one value of r such that $g^m h^r = c$, namely $r = (\alpha - m)/x$. Since r is uniformly random, c is equally likely to be a commitment to any message m.

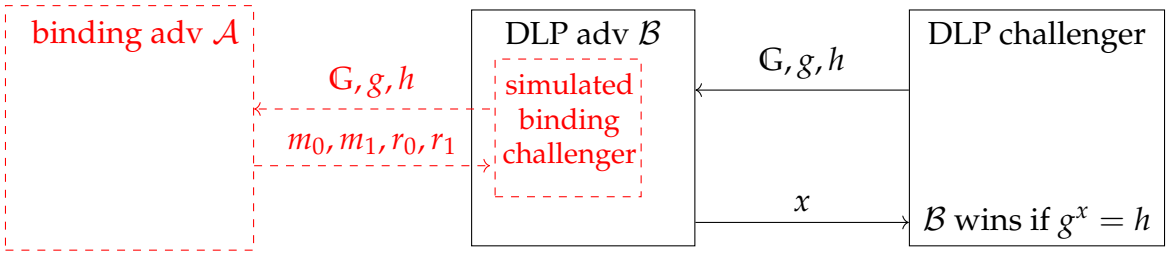
Binding: Let's see how to prove this. We want to prove:

$$\text{DLP is hard} \implies \text{Commitment is binding}$$

Typically, we prove such statements by contrapositive:

$$\text{Commitment is not binding} \implies \text{DLP is easy}$$

We do this via a *hardness reduction:* we assume an adversary $\mathcal{A}$ that can break the binding property, and design an adversary $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine to solve the DLP.



So how do we break a discrete logarithm?

Life advice: If you want to break a discrete log, try to find two different representations of some group element c:

$$g^{m_0} h^{r_0} = c = g^{m_1} h^{r_1}, \qquad m_0 \neq m_1$$
$$g^{m_0} (g^x)^{r_0} = g^{m_1} (g^x)^{r_1}$$
$$m_0 + x \cdot r_0 = m_1 + x \cdot r_1 \bmod q \quad \Longrightarrow \quad \boxed{x = \tfrac{m_1 - m_0}{r_0 - r_1} \bmod q}$$

Given an instance $(\mathbb{G}, g, h)$ of the discrete logarithm problem, we can construct parameters $(\mathbb{G}, g, h)$ for the Pedersen commitment scheme. We then pass these to our binding adversary $\mathcal{A}$, who returns

$$m_0, m_1, r_0, r_1$$

If the binding attacker $\mathcal{A}$ succeeds, then it holds that $g^{m_0} h^{r_0} = g^{m_1} h^{r_1}$ and $m_0 \neq m_1$. Then we can apply the life-advice above to find x. So we can recover the discrete log with the same probability as our attacker $\mathcal{A}$ breaks the binding property. If we assume that the discrete logarithm problem is hard to solve for any PPT adversary, the scheme is computationally binding. □

3 Pseudorandom Functions

We've now already seen two types of security proofs: using the random oracle model, and using a reduction to a computationally hard problem. We're now going to combine the two to build a cool *pseudorandom function*.

Definition 5 (Pseudorandom function)**.** A pseudorandom function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a deterministic, efficiently computable algorithm that takes in a key $k \in \mathcal{K}$ and an input $x \in \mathcal{X}$ and outputs a value $y \in \mathcal{Y}$.

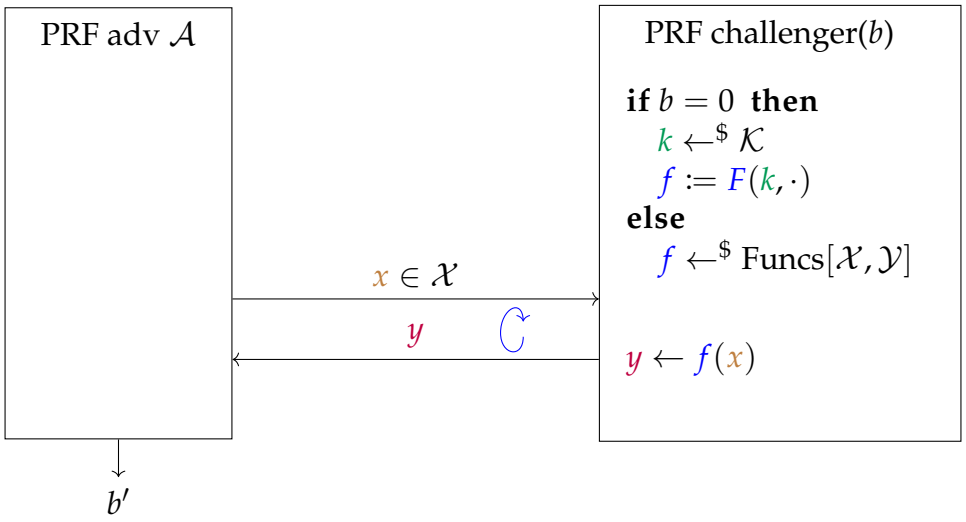
The function is secure if it cannot be distinguished from a truly random function sampled from the set $\text{Funcs}[\mathcal{X}, \mathcal{Y}]$ of all functions from $\mathcal{X}$ to $\mathcal{Y}$. We define the advantage of an adversary $\mathcal{A}$ as

$$\text{AdvPRF}(\mathcal{A}, F) = \left| \Pr[\mathcal{A}^{f(\cdot)} = 1 : f \leftarrow^{\$} \text{Funcs}[\mathcal{X}, \mathcal{Y}]] - \Pr[\mathcal{A}^{F(k,\cdot)} = 1 : k \leftarrow^{\$} \mathcal{K}] \right|$$

where the notation $\mathcal{A}^{f(\cdot)}$ means that $\mathcal{A}$ gets query access to f.

The PRF is secure if any PPT adversary's advantage is negligible.

We can view the PRF security definition as a game between a PRF adversary $\mathcal{A}$ and a challenger. The challenger is given a bit b, and then either samples a random key k for the PRF F or a fully random function and uses this to evaluate queries of the adversary's choice. The adversary wins if it can distinguish which case the challenger is in.



3.1 A Cool PRF

So how do we build a PRF? In fact, we've already done that: the function $H(m, r)$ that we used for the commitment scheme is also a PRF (of the form $F(k, x) = H(k, x)$) in the random oracle model. This is essentially by definition of the random oracle model: for any choice of key k, the function $H(k, \cdot)$ is a random function from $\mathcal{X}$ to $\mathcal{Y}$.

Let's now look at a different PRF which has a number of nice properties. The PRF is $F(k, x) = H(x)^k$ where $H : \mathcal{X} \to \mathbb{G}$ is a hash function that maps to some cyclic group $\mathbb{G}$ of order q.

This PRF appears in a number of cryptographic constructions, some of which we might get to discuss later in the course:

- *Oblivious PRFs*
- *Key-homomorphic PRFs*
- *BLS signatures*

We will prove that this PRF is secure in the random oracle model assuming the DDH assumption holds in $\mathbb{G}$.

Definition 6 (Decisional Diffie-Hellman (DDH) Problem)**.** Let $\mathbb{G}$ be a group of prime order q, with generator g.

The DDH problem is to distinguish between the following two distributions:

$$D_0 = \{(g^\alpha, g^\beta, g^{\alpha\beta}) : \alpha, \beta \leftarrow^{\$} \mathbb{Z}_q\}$$
$$D_1 = \{(g^\alpha, g^\beta, g^\gamma) : \alpha, \beta, \gamma \leftarrow^{\$} \mathbb{Z}_q\}$$

We define the advantage of an adversary $\mathcal{A}$ as

$$\text{AdvDDH}(\mathcal{A}, \mathbb{G}) = |\Pr[W_0] - \Pr[W_1]|$$

where W_0, W_1 are the events that $\mathcal{A}$ outputs 1 when given a sample from D_0 or D_1.

The DDH assumption says that $\text{AdvDDH}(\mathcal{A}, \mathbb{G}) = \text{negl}(\lambda)$ for all PPT adversaries $\mathcal{A}$.

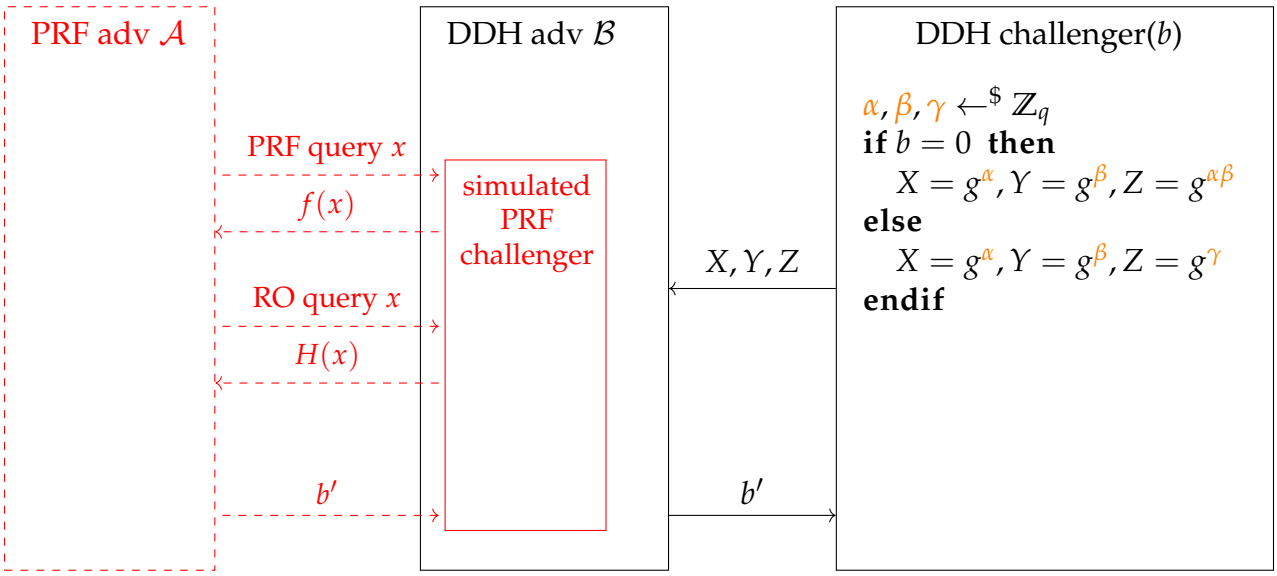
Theorem 3. If the DDH assumption holds in $\mathbb{G}$ and $H : \mathbb{Z}_q \times \mathcal{X} \to \mathbb{G}$ is a random function, then the PRF $F(k, x) = H(x)^k$ is secure. In particular, for every PPT adversary $\mathcal{A}$ that can break the PRF in the RO model, there exists a PPT adversary $\mathcal{B}$ that can solve the DDH problem such that $\text{AdvDDH}(\mathcal{B}, \mathbb{G}) \geq \text{AdvPRF}(\mathcal{A}, F) - \text{negl}(\lambda)$.

You will prove the full theorem in the exercises. Here, we prove a simpler version:

- We restrict ourselves to a PRF adversary $\mathcal{A}$ that makes a *single* PRF query.
- We only show the weaker result that $\text{AdvDDH}(\mathcal{B}, \mathbb{G}) \geq \text{AdvPRF}(\mathcal{A}, F)/Q_{RO}$, where Q_{RO} is the number of RO queries made by $\mathcal{A}$.

Proof. We again proceed by contrapositive: we want to prove that if there exists an adversary $\mathcal{A}$ that breaks our PRF (in the RO model), then we can create an adversary $\mathcal{B}$ that solves the DDH assumption (although only with probability $1/Q_{RO}$).

Here's a picture of what the hardness reduction should look like:



Our adversary $\mathcal{B}$ wants to use $\mathcal{A}$'s ability to break the PRF to solve the DDH problem. For this, $\mathcal{B}$ will simulate a PRF game for $\mathcal{A}$. This means that $\mathcal{B}$ has to respond to $\mathcal{A}$'s PRF queries x with a suitable value $f(x)$.

But $\mathcal{A}$ operates in a world where there exists a random oracle H that it can query. Here's the dirty trick: $\mathcal{B}$ will simulate this random oracle for $\mathcal{A}$, and "cook" the responses to $\mathcal{A}$'s RO queries in such a way that:

- The RO still looks like a random function to $\mathcal{A}$.
- $\mathcal{B}$ can somehow embed the DDH challenge into $\mathcal{A}$'s PRF game.

This is called *programming* the random oracle. Why is this "allowed"? Since we want to show that the PRF is secure in the RO model, the adversary $\mathcal{A}$ cannot make any assumptions about this function and should thus work the same way no matter what the RO is (as long as it's a random function).

Let's see how this works in more detail. Our goal is to program the RO in such a way that when the PRF adversary $\mathcal{A}$ makes a PRF query x, the response they get is either a valid PRF evaluation $H(x)^k$ for some random key k, or a uniformly random element of $\mathbb{G}$. Intuitively, this is what we should use the value Z in the DDH challenge for, since this is either a value $g^{\alpha\beta}$ or a uniformly random element g^{γ} of $\mathbb{G}$.

Let's first assume that $\mathcal{A}$ makes a series of distinct RO queries $x_1, \ldots, x_Q$ and then makes a PRF query on the first message, x_1. We'll set $H(x_1) = X$, and $H(x_i) \leftarrow^{\$} \mathbb{G}$ for $i = 2, \ldots, Q$. This perfectly simulates a random function to $\mathcal{A}$ since X is a uniformly random element of $\mathbb{G}$. Now, when we respond to the PRF query $f(x_1)$ with Z, we have:

$$\begin{aligned} f(x_1) = Z = g^{\gamma} \text{ for } \gamma \leftarrow^{\$} \mathbb{Z}_q & \qquad \text{if } b = 1 \\ f(x_1) = Z = g^{\alpha\beta} = X^{\beta} = H(x_1)^{\beta} = F(\beta, x_1) & \qquad \text{if } b = 0 \end{aligned}$$

So we can perfectly simulate the PRF game for $\mathcal{A}$! So, if $\mathcal{B}$ outputs the same bit b' as $\mathcal{A}$, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins.

Note that by our trick of programming the RO query for x_1, we can then compute $F(\beta, x_1)$ without actually knowing the key β.

We have one problem left: we assumed that $\mathcal{A}$'s PRF query is on the first message x_1 that $\mathcal{A}$ makes to the RO. But $\mathcal{A}$ could of course make its PRF query on any message it wants.

So we'll just *guess* a random index $1 \leq i \leq Q_{RO}$ and set $H(x_i) = X$ and reply to all other RO queries randomly. Then, when $\mathcal{A}$ makes its PRF query, if it is on x_i, we continue the simulation and we output whatever $\mathcal{A}$ outputs. And if it's on any other message, we abort! Thus, $\mathcal{B}$'s advantage is at least $1/Q_{RO}$ times the advantage of $\mathcal{A}$ in the PRF game. □

References

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.