How to Prove Something Secure

1 Introduction

In this document we go over some general strategies for writing a cryptographic proof that a scheme is *secure*, i.e. that it satisfies a particular security definition.

We will give two examples, first proving that a signature scheme satisfies UF-CMA security, and second that a PRG scheme is secure.

2 General Approach

The general approach for proving something secure proceeds as follows.

2.1 Identify the Security Property

You start by identifying exactly what property you are proving that the scheme satisfies. You should be able to frame this definition in the form of a *game* between a challenger and an adversary. Now is the time to identify this game, and notice what the adversary expects to see when it is playing it. Notably, this game is what your *internal adversary* will be playing in every reduction you write throughout the proof.

2.2 Identify the Primitives You Rely On

Next, identify what primitives with what properties you will rely on to do the proof. For example, if the question contains "Let $\Pi = (K, E, D)$ be an IND-CPA encryption scheme ..." and Π is used in the construction of the scheme you are proving secure, you will at some point need to reduce to the IND-CPA security of Π. Make a list of the schemes and properties (like Π and IND-CPA) that you will need to reduce to over the course of the proof.

2.3 Choose Your Proof Technique

Next, you need to identify how your proof will proceed. If you only identified *one* primitive in the previous step, you may be able to do your proof in a single reduction. Otherwise, you will need to do a series of *game hops*. We will focus here on game hopping proofs, as they are the more complex of the two.

If you choose to use game hops, you should next identify your first and last games. What those should be depends on the security property you are proving that your scheme has, specifically whether it is a *distinguishing* property (like PRF or IND-CCA security), or a *search* property (like UF-CMA or collision resistance).

Distinguishing This is the easy case, as your first and last games are given by the property. For example, for PRF security your first game is when the challenger responds with PRF output, and the last is the challenger responding with the output of a random function.

Search In this case, your first game should be one in which your adversary is playing against the scheme as written. Your last game should be one where it is *clearly the case* that finding the search object is not possible. For example, in the case of a MAC scheme M' that is built using a UF-CMA secure MAC scheme M, this may be a game where the challenger simply reponds and verifies signatures using M instead of M'. As M is UF-CMA secure, victory in this game should be impossible.

If you are unable to determine what your final game should be, write one where all the underlying distinguishing primitives are replaced with their ideal version. That is probably it, or very close to it.

2.4 Identifying Game Hops

You should next identify the game hops you will make to move from your first to your last game. As a rule of thumb, there should be one game hop for each *instance* of a cryptographic primitive that has *distinguishing* security. In each game hop you will replace the primitive with its *ideal version*. The ideal version is what the security definition says it is indistinguishable from.

For example, say your construction uses a PRF F, sometimes with key k and sometimes with k', and an encryption scheme (K, E, D). There should be one hop where all PRF invocations with key k are replaced with a random function, and another where all k' invocations are. Finally, there should be a hop where the message input to E is changed (whether you will change this to something specific or just 0^λ is scheme dependent).

Some game hops will not depend on the security of some underlying primitive. These are simple statistical hops, such as replacing $r \oplus m$ with r when r is a uniformly random string and does not appear anywhere else in the output. A list of examples of these types of hops is available at the end of this document.

Finally, you must decide what order you will do your game hops in. A second rule of thumb is that that hops should be ordered by replacing things from the "inside out," replacing the most nested primitives first. If the scheme does something like $E(k, m; F_{k'}(z))$, first replace $F_{k'}$ with a random function, and then replace the message passed to E.

2.5 Justifying Each Hop

Here is where writing the proof actually begins. Write out your list of games, and then for each pair G_i, G_{i+1} prove that:

- The probability that the adversary outputs 0 in G_i is negligibly close to the probability the adversary outputs 0 in G_{i+1} (if you are proving that your scheme satisfies a *distinguishing* definition).
- The probability that the adversary wins in G_i is negligibly close to the probability that the adversary wins in G_{i+1} (if you are proving that your scheme satisfies a *search* definition).

If in your hop you replace a primitive with its ideal version, write a reduction to the security of that primitive. For example, if you replace a PRF F with a random function, you will construct a challenger that wins the PRF game using the adversary that distinguishes between G_i and G_{i+1} as the internal adversary.

If your hop is a simple statistical hop, write out a brief justification for why it is allowed.

If you are proving that your scheme satisfies a distinguishing property, you're done! You've proved that your first game is computationally indistinguishable from your last game.

2.6 Analyzing the Final Game

This step only applies if you are proving that your scheme satisfies a *search* property. You must prove that the probability that an adversary wins in your final game is negligible.

If your final game contains a single search primitive, then this can be done with a single reduction. You will write a reduction to the security of the underlying primitive. For example, if the only primitive left in your final hop is a collision resistant hash function, you will write an adversary that wins the collision resistance game using an adversary that wins your final game as the internal adversary.

If your final game contains *multiple* search primitives (say, a signature scheme and a hash function), then you should split up your adversary's winning chances so you can analyze each individually. Let P_i be the event that your adversary $\mathcal{A}$ wins against primitive P_i. Use the law of probability to deduce that:

$$\begin{aligned}&\mathsf{Pr}[\mathcal{A} \text{ wins}]\\ &= \mathsf{Pr}[\mathcal{A} \text{ wins} \mid P_1]\mathsf{Pr}[P_1] + \mathsf{Pr}[\mathcal{A} \text{ wins} \mid \neg P_1]\mathsf{Pr}[\neg P_1]\end{aligned}$$

And then analyze each probability individually, using a reduction to show that it is negligible, and continue to split for your other primitives.

Once you've proved that the probability of an adversary winning in your final game is negligible, you're done!

3 Example 1: HW 7 Q2

We are told that $(\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Ver})$ is a UF-CMA MAC scheme, and that $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is a IND-CPA secure encryption scheme. We are proving that the below MAC scheme satisfies UF-CMA security.

$\mathsf{KeyGen}'(1^\lambda)$	$\mathsf{Tag}'((r, k_{\mathsf{Tag}}, k_{\mathsf{Enc}}), m)$	$\mathsf{Ver}'((r, k_{\mathsf{Tag}}, k_{\mathsf{Enc}}), m, (\mathsf{ct}, \sigma))$
$r \leftarrow\!\!\$\ \{0,1\}^\lambda$	$\mathsf{ct} \leftarrow \mathsf{Enc}(k_{\mathsf{Enc}}, r)$	**return** $\mathsf{Ver}(k_{\mathsf{Tag}}, m, \sigma) \vee \sigma == r$
$k_{\mathsf{Tag}} \leftarrow \mathsf{KeyGen}(1^\lambda)$	$\sigma \leftarrow \mathsf{Tag}(k_{\mathsf{Tag}}, m)$	
$k_{\mathsf{Enc}} \leftarrow \mathsf{KGen}(1^\lambda)$	**return** (ct, σ)	
return $(r, k_{\mathsf{Tag}}, k_{\mathsf{Enc}})$		

3.1 Identify the Security Property

- We are proving that our scheme satisfies **UF-CMA security**.

3.2 Identify the Primitives You Rely On

Our proof will rely on:

- $(\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Ver})$ and its **UF-CMA** security
- $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ and its **IND-CPA** security.

3.3 Choose Your Proof Technique

- We are relying on multiple primitives, and so we will need to do a series of game hops.
- We are proving that our scheme satisfies a *search* property, and so our first game is the UF-CMA game where the challenger uses $(\mathsf{KeyGen}', \mathsf{Tag}', \mathsf{Ver}')$.
- A good starting point for our last game is one where the encryption scheme is replaced with its "ideal" version. What is the "ideal" version of an encryption scheme? One that encrypts a different message! Therefore, let's start by defining our final game as one identical to the first game, but where ct is instead computed as $\mathsf{Enc}(k_{\mathsf{Enc}}, 0^\lambda)$.

  Next, we want our last game to be "clearly" unwinnable, and so it should be as close as possible to the UF-CMA game played against $(\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Ver})$. We can't get rid of the ct component of the tags, but we can get rid of that "$\vee \sigma == r$" check in Ver'.

  Therefore, our final game is one identical to the first game, with the exception that $\mathsf{ct} \leftarrow \mathsf{Enc}(k_{\mathsf{Enc}}, 0^\lambda)$, and Ver' just returns $\mathsf{Ver}(k_{\mathsf{Tag}}, m, \sigma)$.

- We've identified two changes we want to make, so we just need to choose what order to make them in. It makes sense to remove any information about r from the adversary's view before removing the check for r, so let's do the encryption scheme hop first.

3.4 Identifying Game Hops

- In our first hop we change $\mathsf{ct} \leftarrow \mathsf{Enc}(k_{\mathsf{Enc}}, r)$ to $\mathsf{ct} \leftarrow \mathsf{Enc}(k_{\mathsf{Enc}}, 0^\lambda)$.
- In our second hop we change Ver' to just return $\mathsf{Ver}(k_{\mathsf{Tag}}, m, \sigma)$

3.5 Justifying Each Hop

We are proving a *search* definition, and so need to show that the probability that the adversary wins in each game is negligibly close to the probability that the adversary wins in the next game. We have three games.

- G_0: our first game. the original UF-CMA game.
- G_1: replace ct with $\mathsf{Enc}(k_{\mathsf{Enc}}, 0^\lambda)$
- G_2: change Ver' to just return $\mathsf{Ver}(k_{\mathsf{Tag}}, m, \sigma)$

We must first justify $G_0 \to G_1$. This hop replaces a primitive with an ideal version, and so we must construct a reduction. The primitive we replaced satisfies IND-CPA security, and so we will build an IND-CPA adversary $\mathcal{B}$. We are proving that our scheme satisfies UF-CMA security, and so $\mathcal{B}$ will use an internal adversary $\mathcal{A}$ who's probability of winning the UF-CMA game is non-negligibly different between G_0 and G_1. See the HW7 solutions for the full reduction.

We must next justify $G_1 \to G_2$. This is a statistical hop. The two games are only different if the adversary guesses a λ-bit string that it has no information about, and so the hop is allowed. See the HW7 solutions for more details.

3.6 Analyzing the Final Game

We are proving a search property, and so must do this final step. Our scheme contains a single search primitive, and so we will do this via a reduction.

We are relying on the UF-CMA security of $(\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Ver})$, and so will write an adversary $\mathcal{B}$ that wins the UF-CMA game against it. $\mathcal{B}$ will use an adversary $\mathcal{A}$ that wins our final game as its internal adversary. See the HW7 solutions for the full reduction.

4 Example 2: Miterm I Q1

We are told that $\{F_k\}_{k \in \{0,1\}^\lambda}$ where for all k, $F_k : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ is a secure PRF. We are proving that the below PRG is secure.

$G(s)$
return $F_s(0^\lambda)$

4.1 Identify the Security Property

We are proving that our scheme satisfies **PRG security**.

4.2 Identify the Primitives You Rely On

Our proof will rely on:

- F and its **PRF security**.

4.3 Choose Your Proof Technique

- We are only relying on one primitive, but will still do a series of game hops
- We are proving that our game satisfies a *distinguishing* property. Therefore, our first and last games are set. In the first game the adversary receives $G(s) = F_s(0^\lambda)$ for a random s, and in the second game the adversary receives a uniformly random string.

4.4 Identifying Game Hops

- As we only have one primitive, we don't need any hops in the middle. To sanity check, that hop from the first to the last game does look like replacing $F_s(0^\lambda)$ with its ideal version: the output of a random function at any point is indeed a random string.

4.5 Justifying Each Hop

We are proving a *distinguishing* definition, so we need to prove that the probability that the adversary outputs 0 in each game is negligibly close to the probability that the adversary outputs 0 in the next game. We have two games:

- G_0: our first game. The original PRG game.
- G_1: replace the output with a random string.

We must justify $G_0 \to G_1$. This hop replaces a primitive with an ideal version, and so we must construct a reduction. The primitive we replaced satisfies PRF security, and so we will build a PRF adversary $\mathcal{B}$. We are proving that our scheme satisfies PRG security, and so $\mathcal{B}$ will use an internal adversary $\mathcal{A}$ who's probability of outputting 0 is non-negligibly different between G_0 and G_1. See the Midterm I solutions for the full reduction.

As we are proving a distinguishing property, we're done!

5 Simple Statistical Hops

The following are common statistical game hops.

- Replacing the concatenation of two random strings with a single random string:
$$\left\{ r_0||r_1 \ : \ \begin{array}{c} r_0 \leftarrow\$ \{0,1\}^p \\ r_1 \leftarrow\$ \{0,1\}^q \end{array} \right\} \Leftrightarrow \left\{ r \ : \ r \leftarrow\$ \{0,1\}^{p+q} \right\}$$

- Replacing a generator raised to a random exponent with a random group element:
$$\{ g^x \ : \ x \leftarrow\$ \mathbb{Z}_q \} \Leftrightarrow \{ X \ : \ X \leftarrow\$ G \}$$
(where q is the order of the group)

- For any string m, replacing $m \oplus r$ where r is a uniformly random string with just r, as long as there exists *no* information about r anywhere else in the adversary's view.

- Changing a game where the challenger takes some action if the adversary guesses a random (at least λ-bit) string r to one where the challenger never takes that action, as long as there exists *no* information about r anywhere else in the adversary's view.

- Changing sampling *with* replacement from an exponentially large space to sampling *without* replacement. For example, for any polynomial ℓ:
$$\left\{ (r_0, r_1, \ldots, r_{\ell(\lambda)}) \ : \ \begin{array}{c} r_0 \leftarrow\$ \{0,1\}^\lambda \\ r_1 \leftarrow\$ \{0,1\}^\lambda \\ \cdots \\ r_{\ell(\lambda)} \leftarrow\$ \{0,1\}^\lambda \end{array} \right\}$$
$$\Leftrightarrow$$
$$\left\{ (r_0, r_1, \ldots, r_{\ell(\lambda)}) \ : \ \begin{array}{c} r_0 \leftarrow\$ \{0,1\}^\lambda \\ r_1 \leftarrow\$ \{0,1\}^\lambda \setminus \{r_0\} \\ \cdots \\ r_{\ell(\lambda)} \leftarrow\$ \{0,1\}^\lambda \setminus \{r_0, r_1, \ldots, r_{\ell(\lambda)-1}\} \end{array} \right\}$$

6 Hops That are Never Allowed

- Changing the interface of a scheme. As an example, consider the scheme we prove secure in Section 3. We would not be allowed to make G_1 one in which Tag' is defined as $\mathsf{Tag}'((r, k_{\mathsf{Tag}}, k_{\mathsf{Enc}}), (m, Q, Z))$.

- Changing the shape of the output. You cannot add or remove elements of the output of your scheme functions. For example, for the scheme from

Section 3, you would not be allowed to change the output of Tag' from (ct, σ) to just σ.

Similarly, for a PRF, you would not be able to change the output length of the PRF from $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$.