COL759: Introduction to Cryptography

Venkata Koppula[*]

July 24th, 2023

Contents

[*]kvenkata@cse.iitd.ac.in

Many thanks to Ramprasad Saptarishi for this LaTeX template.

1 Course Introduction

Lecture 1: July 25th, 2023

Welcome to COL759! The objective of this course is to understand the foundational building blocks of cryptography, and see how they can used to obtain provably secure cryptosystems. This will usually involve four steps:

1. Giving a formal security definition
2. Identifying appropriate building blocks
3. Proposing a construction based on these building blocks
4. Showing that the proposed construction satisfies the formal security definition

There could be multiple formal definitions for the same 'intuitive' security objective. For instance, if we consider private key encryption, what does it mean for an encryption scheme to be secure? How do we compare different security definitions?

Why provable security: we saw a few scenarios in the first lecture, where 'intuitively secure' constructions are not provably secure.

- The first example involved the Data Encryption Standard (DES). This was a popular encryption standard in the 70s. This had 56 bit secret keys. Today, cryptosystems with 56-bit keys can be broken in $\approx 2^{56}$ steps using exhaustive key search. Therefore, a natural idea is the following: choose a 112-bit key, and break it into two 56-bit keys k_1 and k_2. First encrypt the message using k_1, and then encrypt the resulting ciphertext using k_2. Exhaustive key search is not feasible any more (since 2^{112} is infeasible), however there's a clever attack that works in approximately 2^{56} steps! This attack does not use any structure of DES (and as a result, it works for any encryption scheme).

  A natural next question is whether triple encryption would work. And one can prove that triple-encryption is better than single-encryption or double-encryption (and therefore 3DES is provably better than DES or 2DES).

  In class, we saw that there's a sophisticated 2^{44} time attack that works for DES, but doesn't work for 2DES. In this sense, 2DES is slightly better than DES.

- The AES circuit can be used to encrypt 128 bit messages. How do we use it to encrypt longer messages?

  The first natural idea is to decompose the message into blocks of size 128 each, and then apply AES to each of them separately. This is a bad idea (referred to as 'naive' approach below, and leads to several attacks in real-world scenarios). For instance, in Figure 1, we have an encryption of a popular image using this approach. Can you identify the image?

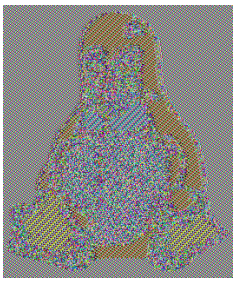


Figure 1: Image encrypted using 'naive' approach

A couple of interesting ideas were proposed in class:

- Use the ciphertext generated from one block to 'mask' the next message block, and then apply AES to this.
- Use the ciphertext generated from one block as the AES for the next block.

Both these are better than the 'naive' approach. While these approaches should also not be used in practice (this will become clear when we see the formal security definition for encryption), they do offer some security.

Throughout this course, we will see several security proofs. We will also see a few attacks which happened because of one of the following mistakes:

- the building blocks were not used properly. For instance, the key should have been used only once, but it was re-used. Or maybe the keys were supposed to be chosen uniformly at random each time, but there were some correlations in the keys.
- some modifications were made to the building blocks, resulting in a totally broken system. Note that these building blocks are very brittle.
- the building blocks were not 'put-together' properly. This is another reason why we need provable security. Just because we used two secure building blocks doesn't mean that the resulting system is secure.
- finally, there were attacks resulting from underestimation of the adversary's power.

Note that these are not implementation flaws, these are design flaws, and these could have been avoided by first choosing an appropriate threat model, and then proving security in this threat model. This is what we will aim to do in this course.

2 Perfect Security

In this section, we will discuss the most basic, and the most secure, private key encryption scheme — Shannon's encryption scheme. First, let us set up the syntax for a private key encryption scheme. A private key encryption scheme $\mathcal{E}$ with key space $\mathcal{K}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ consists of two algorithms $(\mathsf{Enc}, \mathsf{Dec})$ with the following properties:

- $\mathsf{Enc}\,(k \in \mathcal{K}, m \in \mathcal{M})$: The encryption algorithm takes as input a key $k \in \mathcal{K}$, message in $\mathcal{M}$ and outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$. The encryption algorithm can either be deterministic (in which case we write $\mathsf{ct} = \mathsf{Enc}\,(k, m)$) or can be randomized (in which case we write $\mathsf{ct} \leftarrow \mathsf{Enc}(k, m)$, or $\mathsf{ct} = \mathsf{Enc}\,(k, m; r)$ in case we want to make the encryption randomness r explicit).
- $\mathsf{Dec}\,(k \in \mathcal{K}, \mathsf{ct} \in \mathcal{C})$: The decryption algorithm takes as input a key k, ciphertext $\mathsf{ct} \in \mathcal{C}$, and either outputs a message $m \in \mathcal{M}$, or a special symbol $\perp$ (indicating that decryption failed). We will assume the decryption algorithm is deterministic.

We want these algorithms to be efficient (that is, the running time should be polynomial in the input length). Additionally, the algorithms must satisfy *correctness* and *security*. Correctness is easy to define, so let's start with that.

Correctness: For every key $k \in \mathcal{K}$, and message $m \in \mathcal{M}$,

$$\Pr\left[\mathsf{Dec}\,(k, \mathsf{Enc}\,(k, m)) = m\right] = 1.$$

Here, the probability is over the randomness used for encryption.

Security Definition: Shannon, in 1945, gave the following definition for perfect one-time secure encryption schemes.

Definition 2.1. *An encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$, key space $\mathcal{K}$ and ciphertext space $\mathcal{C}$ is said to be perfect one-time secure if for all messages $m_0, m_1 \in \mathcal{M}$, all ciphertexts $\mathsf{ct} \in \mathcal{C}$,*

$$\Pr_k\left[\mathsf{ct} \leftarrow \mathsf{Enc}(k, m_0)\right] = \Pr_k\left[\mathsf{ct} \leftarrow \mathsf{Enc}(k, m_1)\right].$$

Here the probability is over the choice of key k, and the randomness used for encryption. ◊

A natural question that came up in class: what's the intuition behind this definition? We will address this in the next lecture by giving an equivalent definition that captures our intuition better (in Section 2.4).

First, as noted by one of the students in class, this definition (and any definition of secure encryption schemes) can hold only if the two messages have the same length. This is because *the ciphertext cannot hide the message length*, irrespective of how strong the encryption scheme is. In Assignment 1, we will see how this was exploited for some real-world security breaches.

2.1 *Perfect One-Time Security - Examples*

Let us recall a few historic encryption schemes (with bounded message space) and check whether they satisfy Shannon's one-time security.

Construction 2.2 (Substitution Cipher)**.** *Let* $\mathcal{M} = \mathcal{C} = [a-z]^{100}$*, and* $\mathcal{K} = \{\sigma : [a-z] \to [a-z]\}$*. The substitution cipher is defined as follows:*

- $\mathsf{Enc}(k = \sigma, m = (m_1\ m_2\ \ldots\ m_{100})) = (\sigma(m_1)\ \sigma(m_2)\ \ldots\ \sigma(m_{100}))$
- $\mathsf{Dec}(k = \sigma, \mathsf{ct} = (\mathsf{ct}_1\ \mathsf{ct}_2\ \ldots\ \mathsf{ct}_{100})) = (\sigma^{-1}(\mathsf{ct}_1)\ \sigma^{-1}(\mathsf{ct}_2)\ \ldots\ \sigma^{-1}(\mathsf{ct}_{100}))$

◊

This scheme does not satisfy Definition 2.1. Consider $m_0 = (a\ a\ \ldots\ a)$ and $m_1 = (a\ b\ \ldots\ b)$, and $\mathsf{ct} = (c\ c\ \ldots\ c)$. Note that $\mathsf{Pr}_k[\mathsf{Enc}(k, m_0)] = 1/26$, but $\mathsf{Pr}_k[\mathsf{Enc}(k, m_1) = 0]$.

Next, let us consider a variant of Vigenère's encryption scheme, where we allow the key size to be as large as the message space.

Construction 2.3 (Modified Vigenère Cipher)**.** *Let* $\mathcal{M} = \mathcal{C} = [a-z]^{100}$*, and* $\mathcal{K} = \{(\sigma_1, \ldots, \sigma_{100})$ *where* $\sigma_i : [a-z] \to [a-z]\}$*. The modified Vigenère cipher is defined as follows:*

- $\mathsf{Enc}(k = (\sigma_1\ \sigma_2\ \ldots\ \sigma_{100}), m = (m_1\ m_2\ \ldots\ m_{100})) = (\sigma_1(m_1)\ \sigma_2(m_2)\ \ldots\ \sigma_{100}(m_{100}))$
- $\mathsf{Dec}(k = (\sigma_1\ \sigma_2\ \ldots\ \sigma_{100}), \mathsf{ct} = (\mathsf{ct}_1\ \mathsf{ct}_2\ \ldots\ \mathsf{ct}_{100})) = (\sigma_1^{-1}(\mathsf{ct}_1)\ \sigma_2^{-1}(\mathsf{ct}_2)\ \ldots\ \sigma_{100}^{-1}(\mathsf{ct}_{100}))$

◊

Check that this construction satisfies Shannon's one-time perfect security. For any message $m \in [a-z]^{100}$ and ciphertext $\mathsf{ct} \in [a-z]^{100}$, $\mathsf{Pr}[\mathsf{ct} = \mathsf{Enc}(k, m)] = 1/26^{100}$.

Note: The Vigenère cipher proposed in the 1500s (that was broken in the 1800s) had a bounded number of permutations in the secret key, and was used to encrypt unbounded length messages. As a result, the scheme was not perfect one-time secure. Check that the attack used against Construction 2.2 also works here.

Shannon's One-Time Pad There is a much simpler scheme that satisfies Definition 2.1, this is popularly referred to as Shannon's One-Time Pad.

Construction 2.4 (Shannon's One-Time Pad)**.** *Let* $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$*, where n denotes the length of message to be encrypted.*

- $\mathsf{Enc}(k, m) = k \oplus m$ *where* $\oplus$ *denotes bit-wise XOR.*
- $\mathsf{Dec}(k, \mathsf{ct}) = k \oplus \mathsf{ct}$.

◊

Check that this scheme is correct, and for any message m and ciphertext $\mathsf{ct} \in \{0,1\}^n$, $\mathsf{Pr}_k[\mathsf{Enc}(k,m) = \mathsf{ct}] = 1/2^n$.

Unfortunately, both Construction 2.3 and Construction 2.4 have very large secret keys (as large as the message length). This is a problem with perfect security in general. If a scheme is correct and $|\mathcal{K}| < |\mathcal{M}|$, then the scheme cannot satisfy Definition 2.1.

2.2 *Perfect One-Time Security: Lower Bound*

Let us make a simple observation about encryption schemes that satisfies Definition 2.1.

Observation 2.5. *Let m_0, m_1 be two distinct messages, and c be a ciphertext. Suppose there are exactly t secret keys $k_1, \ldots, k_t$ such that $\mathsf{Enc}(k_i, m_0) = c$, for $i \in [1,t]$. Then there must exist t keys $k'_1, \ldots, k'_t$ such that $\mathsf{Enc}(k'_j, m_1) = c$, for $j \in [1,t]$. Moreover, for all $i, j \in [t]$, $k_i \neq k'_j$.*

Proof. Consider an encryption schemes that satisfies Definition 2.1. We have for all m_0, m_1, c,

$$\mathsf{Pr}_k[\mathsf{Enc}(k,m_0) = c] \;=\; \mathsf{Pr}_k[\mathsf{Enc}(k,m_1) = c].$$

Observe that $\mathsf{Pr}_k[\mathsf{Enc}(k,m_0) = c] = t/|\mathcal{K}|$. Thus, $\mathsf{Pr}_k[\mathsf{Enc}(k,m_1) = c]$ must also be $t/|\mathcal{K}|$, thereby proving that there are exactly t keys say $k'_1, \ldots, k'_t \in \mathcal{K}$ that maps m_1 to c.

The second part follows from perfect correctness. Suppose $k_i = k'_j$ for some $i, j \in [1,t]$. Then $\mathsf{Dec}(k_i, c) = \mathsf{Dec}(k'_j, c)$, implying m_0 must be equal to m_1. This contradicts the assumption that m_0, m_1 are distinct messages. □

Now, fix any message m^* and key k^*, and let $c^* = \mathsf{Enc}(k^*, m^*)$. There is at least one key mapping m^* to c^*. Therefore, using the observation above, for any message m, there exists a key, say $k_m \in \mathcal{K}$, such that $\mathsf{Enc}(k_m, m) = c^*$. Moreover, we showed above that for any distinct messages $m \neq m'$, the corresponding keys k_m and $k_{m'}$ are distinct. Using this, we can conclude that $|\mathcal{K}| \geq |\mathcal{M}|$.

2.3 *Other limitations of One Time Pad*

Lecture 2: July 28th, 2023

Below we list a few other limitations of the one-time pad.

- **Two time pad attack:** Let c_1 and c_2 be two ciphertexts obtained via the one time pad encryption scheme **using the same secret key**, say k. An adversary, given two ciphertexts c_1 and c_2, can learn the XOR of the underlying messages. If $c_1 = \mathsf{Enc}(m_1, k) = m_1 \oplus k$ and $c_2 = \mathsf{Enc}(m_2, k) = m_2 \oplus k$, then $c_1 \oplus c_2 = m_1 \oplus m_2$.

  Similar to one-time perfect security, we can defined two-time perfect security. For every $(m_{00}, m_{01}), (m_{10}, m_{11}) \in \mathcal{M}^2$ and every ciphertext pair $(c_0, c_1) \in \mathcal{C}^2$, we require that

  $$\mathsf{Pr}_k[\mathsf{Enc}(k,m_{00}) = c_0 \wedge \mathsf{Enc}(k,m_{01}) = c_1] = \mathsf{Pr}_k[\mathsf{Enc}(k,m_{10}) = c_0 \wedge \mathsf{Enc}(k,m_{11}) = c_1].$$

  Check that Shannon's one-time pad does not satisfy this definition.

Exercise 2.1. *Does Construction 2.3 (or some other scheme) satisfy this definition?*

- **Malleability attack:** Given an encryption of a message m using a key k, the adversary can modify the ciphertext so that the decryption using k outputs $f(m)$. That is, suppose you encrypt your hard disk with a key k, and the resulting ciphertext is ct. The adversary does not know anything about the contents of your hard disk, but can modify the ciphertext so that (a) you will not detect any tampering (b) worse, when you decrypt the tampered ciphertext using your key k, you might receive $f(m)$. The tampering will depend on the function f, but not on the message or the secret key.

  Note that the adversary knows neither m nor k, but is able to modify the underlying message. For example, if the message and key are both n bits long, the adversary can flip all (or a fraction of) the message bits by XORing the ciphertext with an appropriate n bit string.

Exercise 2.2. *Do these (or similar) malleability attacks also work against modified Vigenère scheme (Construction 2.3)?*

Challenge Problem 1. *Are malleability attacks also inherently present in any perfectly secure encryption scheme? Or, can we have an encryption scheme that is perfect one-time secure, and does not have any malleability attacks?* [a]

[a] we haven't defined 'malleability attacks' formally, we will do this later in the course. In case you have a candidate construction that doesn't seem to have any malleability attacks, you can think about a reasonable definition for capturing malleability attacks, or discuss with me after class.

2.4 *An Equivalent (and More Intuitive?) Definition*

Any cryptographic primitive's security definition starts with an informal understanding of what information the adversary gets, and what the adversary aims to do. For now, we limit ourselves to the most basic setting - the adversary receives exactly one ciphertext, and must learn something about the underlying message. Note that this statement is informal for various reasons. First, with regard to the information that the adversary gets: how is the ciphertext generated? Second, with regard to the adversary's objective: what does it mean to '*learn something about the underlying message*'? These things are formally captured via a *security game*. Throughout this course, we will see several security games, and often, there will be multiple different possible security games (for the same informal threat description).

A security game is defined via interaction between a *challenger* and an *adversary*. At the end, there is a *winning condition* for the adversary, and informally, we say that a scheme is secure if no adversary can win the security game with noticeable probability. Let us see a few examples for security games.

The first two examples will be for capturing the following intuition : the adversary, given one ciphertext, hides the underlying message. Now, 'hides the message' could itself mean several things - maybe the adversary should not learn anything about the message? Or maybe the adversary should not learn the entire message? We will start with the former, and the security game is given in Figure 2.

Semantic Security Game

Intuition: Adversary, given one ciphertext, does not learn anything about underlying message

1. Adversary sends two messages m_0, m_1 of equal length.
2. Challenger picks a uniformly random key $k \leftarrow \mathcal{K}$, a uniformly random bit b (which denotes the message to be encrypted) and sends $\mathsf{ct} \leftarrow \mathsf{Enc}(k, m_b)$.
3. Adversary outputs its guess b', and wins the game if $b = b'$.

Figure 2: The semantic security game is defined with respect to an encryption scheme $\mathcal{E}$ and adversary $\mathcal{A}$

Clearly, the adversary can win this game with probability 1/2 (by sending a uniformly random bit as its final output). For a perfect encryption scheme, no adversary should do better than a random guess. This is formally stated below.

Definition 2.6. *An encryption scheme $\mathcal{E}$ satisfies perfect semantic security if, for all adversaries $\mathcal{A}$, $\Pr[\mathcal{A}$ wins Semantic Security Game w.r.t. $\mathcal{E}] = 1/2$.* ◊

Similarly, we can also define a security game for capturing the following intuition: given an encryption of a random message using a random key, adversary should not learn the entire message. This game is given in Figure 3.

Message Recovery Security Game

Intuition: Adversary, given an encryption of a random message, does not learn the underlying message

1. Challenger picks a uniformly random key $k \leftarrow \mathcal{K}$, a uniformly random message $m \leftarrow \mathcal{M}$ and sends $\mathsf{ct} \leftarrow \mathsf{Enc}(k, m)$.
2. Adversary outputs its guess m', and wins the game if $m = m'$.

Figure 3: The message recovery security game is defined with respect to an encryption scheme $\mathcal{E}$ and adversary $\mathcal{A}$

Note that the adversary can always win the message guessing security game (defined in Figure 3) with probability $1/|\mathcal{M}|$. Naturally, we want our encryption scheme to be such that the adversary cannot do better than $1/|\mathcal{M}|$.

Definition 2.7. *An encryption scheme $\mathcal{E}$ satisfies perfect security against message recovery if, for any adversary $\mathcal{A}$, $\Pr[\mathcal{A}$ wins the Message Recovery Security Game w.r.t. $\mathcal{E}] = \frac{1}{|\mathcal{M}|}$.* ◊

Now that we have a few different security definitions for (one-time) encryption security, a natural question is to study the relationship between these definitions. For example, we have an easy-to-work-with definition for perfect

security (Definition 2.1), and an intuitive definition for perfect security (Definition 2.6). It is not too difficult to show that these two definitions are equivalent. Here, we will only prove one direction. We will show that Definition 2.6 implies 2.1.

Observation 2.8. *If an encryption scheme $\mathcal{E}$ (with deterministic encryption) satisfies Definition 2.6, then it also satisfies Definition 2.1.*

Proof. We will work with the contrapositive - if an encryption scheme does not satisfy Definition 2.1, then it does not satisfy Definition 2.6.

Suppose an encryption scheme does not satisfy Shannon's definition. Then there exist two messages m_0^*, m_1^* and a ciphertext ct^* such that $\mathsf{Pr}\left[\mathsf{ct}^* = \mathsf{Enc}(k, m_0^*)\right] > \mathsf{Pr}\left[\mathsf{ct}^* = \mathsf{Enc}(k, m_1^*)\right]$. Let $p_b = \mathsf{Pr}\left[\mathsf{ct}^* = \mathsf{Enc}(k, m_b^*)\right]$. We will construct an adversary that wins the semantic security game with non-zero probability. The adversary knows m_0^*, m_1^* and ct^*. It sends (m_0^*, m_1^*) to the challenger, and receives a ciphertext ct. If $\mathsf{ct}^* = \mathsf{ct}$, then adversary sends $b' = 0$ as its guess. Else, if $\mathsf{ct}^* \neq \mathsf{ct}$, the adversary simply sends a uniformly random bit b' as its guess.

$$
\begin{aligned}
&\mathsf{Pr}\left[\mathcal{A} \text{ wins w.r.t. } \mathcal{E}\right] \\
=&\mathsf{Pr}\left[\text{Challenger chooses 0, and } \mathcal{A} \text{ outputs 0}\right] + \mathsf{Pr}\left[\text{Challenger chooses 1, and } \mathcal{A} \text{ outputs 1}\right] \\
=&\mathsf{Pr}\left[\text{Challenger chooses 0}\right]\mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 0}\right] \\
&+ \mathsf{Pr}\left[\text{Challenger chooses 1}\right]\mathsf{Pr}\left[\mathcal{A} \text{ outputs 1} \mid \text{Challenger chooses 1}\right] \\
=&\frac{1}{2}\mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 0}\right] + \frac{1}{2}\mathsf{Pr}\left[\mathcal{A} \text{ outputs 1} \mid \text{Challenger chooses 1}\right] \\
=&\frac{1}{2}\mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 0}\right] + \frac{1}{2}\left(1 - \mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 1}\right]\right) \\
=&\frac{1}{2} + \frac{1}{2}\left(\mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 0}\right] - \mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 1}\right]\right)
\end{aligned}
$$

It suffices to show that

$$\mathsf{AdvPerfSS} = \mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 0}\right] - \mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses 1}\right]$$

is positive. This term will often arise in our analysis, and is referred to as the *advantage of $\mathcal{A}$ with respect to $\mathcal{E}$ in the perfect semantic security game*.

Let $\mathcal{K}[m_b^*, \mathsf{ct}^*] = \{\, k : \mathsf{Enc}(k, m_b^*) = \mathsf{ct}^* \,\}$ (the set of all keys that map m_b^* to ct^*). Note that $\mathcal{K}[m_0^*, \mathsf{ct}^*] \cap \mathcal{K}[m_1^*, \mathsf{ct}^*] = \varnothing$ and $|\mathcal{K}[m_b^*, \mathsf{ct}^*]| = p_b \cdot |\mathcal{K}|$.

$$
\begin{aligned}
&\mathsf{Pr}\left[\mathcal{A} \text{ outputs 0} \mid \text{Challenger chooses } b\right] \\
=&\mathsf{Pr}\left[\begin{array}{c}\text{Challenger chooses } k \in \mathcal{K}[m_b^*, \mathsf{ct}^*] \\ \mathcal{A} \text{ outputs 0}\end{array} \;\middle|\; \text{Challenger chooses } b\right] \\
&+ \mathsf{Pr}\left[\begin{array}{c}\text{Challenger chooses } k \notin \mathcal{K}[m_b^*, \mathsf{ct}^*] \\ \mathcal{A} \text{ outputs 0}\end{array} \;\middle|\; \text{Challenger chooses } b\right]
\end{aligned}
$$

First, note that

$$\begin{aligned}
&\mathsf{Pr}\left[\begin{array}{c}\text{Challenger chooses } k \notin \mathcal{K}[m_b^*, \mathsf{ct}^*] \\ \mathcal{A} \text{ outputs } 0\end{array} \;\middle|\; \text{Challenger chooses } b\right] \\
=&\frac{1}{2} \cdot \mathsf{Pr}\left[\text{ Challenger chooses } k \notin \mathcal{K}[m_b^*, \mathsf{ct}^*] \;\middle|\; \text{Challenger chooses } b\right] \\
=&\frac{1}{2} \cdot (1 - p_b)
\end{aligned}$$

This is because in this case, the adversary receives a ciphertext not equal to ct^*, and therefore it sends a random guess.

Next, note that the probability

$$\mathsf{Pr}\left[\begin{array}{c}\text{Challenger chooses } k \in \mathcal{K}[m_b^*, \mathsf{ct}^*] \\ \mathcal{A} \text{ outputs } 0\end{array} \;\middle|\; \text{Challenger chooses } b\right]$$

is equal to p_b. This is because the adversary outputs '0' when it receives ct^*. Putting everything together, we get that

$$\begin{aligned}
\mathsf{Pr}\left[\mathcal{A} \text{ wins w.r.t. } \mathcal{E}\right] &= \frac{1}{2} + \frac{1}{2}\mathsf{AdvPerfSS} \\
&= \frac{1}{2} + \frac{1}{2}\left(p_0 + \frac{1}{2}(1 - p_0) - p_1 - \frac{1}{2}(1 - p_1)\right) \\
&= \frac{1}{2} + \frac{(p_0 - p_1)}{4}.
\end{aligned}$$

Since $p_0 > p_1$, the encryption scheme does not satisfy Definition 2.6. □

Exercise 2.3. *Suppose you know that an encryption scheme $\mathcal{E}$ does not satisfy Definition 2.1; that is, there exist two messages m_0^*, m_1^* and a ciphertext ct^* such that $\mathsf{Pr}\left[\mathsf{Enc}(k, m_0^*) = \mathsf{ct}^*\right] = p_0$, $\mathsf{Pr}\left[\mathsf{Enc}(k, m_1^*) = \mathsf{ct}^*\right] = p_1$ and $p_0 > p_1$. Consider the following adversary $\mathcal{A}$ in the perfect semantic security game (Figure 2): the adversary sends m_0, m_1 as the challenge messages, and receives a ciphertext ct. The adversary always guesses '0'. Is this adversary always guaranteed to win with probability strictly greater than $1/2$?* [a]

[a] Thanks to one of the students who asked this question after class today.

Exercise 2.4. *Is Definition 2.6 equivalent to Definition 2.7? Either prove equivalence, or show a scheme that satisfies one definition, but not the other one.*

An immediate consequence of Observation 2.8 is that if an encryption scheme satisfies Definition 2.6, then the key space must be at least as large as the message space. Can we relax Definition 2.6 to allow encryption schemes with short keys but large messages?

The first suggestion in class was to relax the strict requirement on the winning probability.

Definition 2.9. *An encryption scheme $\mathcal{E}$ satisfies ϵ-perfect semantic security if, for all adversaries $\mathcal{A}$,* $\Pr[\mathcal{A}$ *wins Semantic Security Game w.r.t.* $\mathcal{E}] \leq 1/2 + \epsilon$. ◊

One can show that this relaxation is not very helpful — any encryption scheme satisfying this definition must have key space $\mathcal{K}$ of size at least $(1 - 2\epsilon)|\mathcal{M}|$.

The next suggestion was to restrict our attention to only *efficient* adversaries - adversaries whose running time is polynomially bounded. Just restricting the adversary's running time is not enough. Let $\mathcal{M} = \{0,1\}^*$, and $\mathcal{K} = \{0,1\}^n$. Consider the following (polynomial time) adversary: it picks two random messages m_0, m_1 of equal length (say $100 \cdot n$ bits long) and sends them to the challenger. The challenger picks one of them at random, picks a uniformly random key k and sends $\mathsf{ct} = \mathsf{Enc}(k, m_b)$. The adversary then picks a uniformly random key k', decrypts ct. If the resulting decryption is either m_0 or m_1, then the adversary sends its guess accordingly. Otherwise, the adversary sends a uniformly random bit as its guess. If $k = k'$ (and this happens with probability $1/2^n$), then the adversary definitely wins. If $k \neq k'$, then the adversary's winning probability is very close to 1/2. Therefore, overall, one can show that the winning probability of this adversary is strictly greater than 1/2 (albeit only very slightly greater than 1/2). [1]

This brings us to the final idea (and one that works): restrict attention to polynomial-time adversaries, and allow the adversary to win with probability slightly greater than 1/2.

Informally, an encryption scheme satisfies semantic security if for any *efficient* adversary, the winning probability is at most 1/2 + *negligible quantity*. We will make these formal in the next chapter.

Chapter Summary and References

- We started with Shannon's definition of perfect one-time (and more generally, t-time) security.
- Next, we saw a few schemes that satisfy perfect one-time security.
- After this, we saw that any encryption scheme satisfying perfect one-time security must have key space as large as the message space.
- We then introduced the perfect semantic security definition (which is closer to our intuitive understanding of a one-time perfectly secure encryption scheme), and argued that this definition is equivalent to Shannon's definition of perfect one-time security. Looking ahead, in the next section, we will try to put reasonable restrictions on this definition in order to achieve encryption schemes with key space much smaller than the message space.
- The one-time pad does not offer two-time security. Additionally, it is also susceptible to 'malleability attacks'.

Textbook References: Boneh-Shoup (v6), Section 2.1.

[1] The formal probability analysis is a bit subtle, and you are encouraged to work out the details. Discuss with me after class if you are stuck.

3 Stream Ciphers and One-Time Security

Lecture 3: August 1st, 2023

We ended the last section/lecture with an alternate definition for Shannon's definition of one-time security. We also informally discussed how to relax the definition to achieve one-time security with succinct encryption keys. An encryption scheme is (one-time) secure if any *efficient* adversary can win the one-time semantic security game with probability at most 1/2+*negligible*. In this section, we will first make these terms formal. Informally, 'efficient' refers to polynomial-time computation, and 'negligible' refers to something that's smaller than inverse-polynomial.

3.1 *Enter the security parameter*

The security parameter determines how secure you want your cryptosystem to be.

- All algorithms in the cryptosystem must run in time polynomial in the security parameter.
- The key space must grow with the security parameter.
- The adversary is allowed to be randomized, and can run in time polynomial in the security parameter.
- In the case of encryption security, we want the adversary's winning probability to be less than $1/2 + 1/p(\text{security parameter})$ for any polynomial p. Functions which grow slower than inverse-polynomial are called negligible functions.

First, let us define negligible functions formally.

Definition 3.1. *A function $\mu : \mathbb{N} \to [0,1]$ is called negligible if, for any polynomial p, there exists $n_0 \in \mathbb{N}$ such that for all $n > n_0$, $\mu(n) < 1/p(n)$.* ◊

Examples of negligible functions:

- $1/2^n$
- $1/n^{\log n}$
- $p(n)/n^{\log n}$ for any fixed polynomial p.

If μ is a negligible function, and p is a polynomial, then note that $\mu(n) \cdot p(n)$ is also negligible. We will use this observation several times in this course.

3.2 *Formal Definitions for Computational Security*

Since perfect security (with short keys) is impossible to achieve, we will focus on security against polynomial-time adversaries. This is referred to as *computational security*.

Semantic Security

Let us go back to the semantic security game (Figure 2). The adversary sends two messages, the challenger picks one of them and encrypts it (using a random key), and the adversary must guess which message was encrypted. First, note that the winning probability of the adversary is now a function of the security parameter (since this game is played with respect to some security parameter). Let's denote this function as $p_{\mathcal{A}}$. We want $|p_{\mathcal{A}} - 1/2|$ to be bounded by a negligible function. The formal definition is given below.

Definition 3.2. *An encryption scheme $\mathcal{E}$ is said to be semantically secure if, for any probabilistic polynomial time (p.p.t.) adversary $\mathcal{A}$,*

$$p_{\mathcal{A}}(\lambda) = \left| \mathsf{Pr}\left[\mathcal{A} \text{ wins the semantic security game w.r.t. } \mathcal{E}\right] - \frac{1}{2} \right|$$

is a negligible function. ◊

Often, when we analyse the probability of an adversary succeeding, we proceed as follows:

$$\begin{aligned}
&\mathsf{Pr}\left[\mathcal{A} \text{ wins the semantic security game w.r.t. } \mathcal{E}\right] \\
=&\mathsf{Pr}\left[\begin{array}{l}\text{Challenger picks bit } b \\ \mathcal{A} \text{ sends } b \text{ as its guess}\end{array}\right] \\
=&\frac{1}{2}\left(\mathsf{Pr}\left[\mathcal{A} \text{ sends } 0 \mid \text{ Challenger picks } 0\right]\right) \\
&+\frac{1}{2}\left(\mathsf{Pr}\left[\mathcal{A} \text{ sends } 1 \mid \text{ Challenger picks } 1\right]\right) \\
=&\frac{1}{2}\left(\mathsf{Pr}\left[\mathcal{A} \text{ sends } 0 \mid \text{ Challenger picks } 0\right]\right) \\
&+\frac{1}{2}\left(1 - \mathsf{Pr}\left[\mathcal{A} \text{ sends } 0 \mid \text{ Challenger picks } 1\right]\right)
\end{aligned}$$

As a result, in order to show that an encryption scheme is semantically secure, it suffices to show that

$$\left|\mathsf{Pr}\left[\mathcal{A} \text{ sends } 0 \mid \text{ Challenger picks } 0\right] - \mathsf{Pr}\left[\mathcal{A} \text{ sends } 0 \mid \text{ Challenger picks } 1\right]\right|$$

is a negligible function. This quantity will often arise in our analysis, and it is referred to as the 'advantage' of $\mathcal{A}$ w.r.t. the encryption scheme $\mathcal{E}$ in the semantic security game.

Security against Message Recovery Attacks

We can similarly define security against message recovery attacks. Recall, the challenger in this security game picks a uniformly random message, and sends its encryption. The adversary's goal is to guess this message. In perfect security against message recovery attacks, we require this probability to be exactly $1/|\mathcal{M}|$, but here we allow the adversary to win with probability negligibly close to $1/|\mathcal{M}|$.

Definition 3.3. *An encryption scheme $\mathcal{E}$ is said to be secure against message recovery*

attacks if for any p.p.t. adversary $\mathcal{A}$,

$$\left| \mathsf{Pr}\left[\mathcal{A} \textit{ wins the message recovery game w.r.t. } \mathcal{E}\right] - \frac{1}{|\mathcal{M}|} \right|$$

is a negligible function. ◊

Relationship between the above definitions

When given multiple security definitions, a natural question to ask is which one is 'better'. Sometimes, definitions may be incomparable, but in this case, we will have a clear answer - semantic security is strictly better than security against message recovery. We will show that Definition 3.2 implies Definition 3.3, but the other direction does not hold true. In other words, we will show that if an encryption scheme $\mathcal{E}$ does not satisfy security against message recovery (that is, there exists a p.p.t. adversary $\mathcal{A}$ that wins the message-recovery game w.r.t. $\mathcal{E}$), then $\mathcal{E}$ does not satisfy semantic security (that is, there exists a p.p.t. algorithm $\mathcal{B}$ that wins the semantic security game w.r.t. $\mathcal{E}$). This algorithm $\mathcal{B}$ is called a *reduction*.

This will be our first (of many) security proofs that we will see in the course; please go through this in detail.

Claim 3.4. *If there exists a p.p.t. adversary $\mathcal{A}$ that wins the message recovery game w.r.t. $\mathcal{E}$ with probability $1/|\mathcal{M}| + \epsilon$, then there exists a p.p.t. algorithm $\mathcal{B}$ such that* $\mathsf{SSAdv}[\mathcal{B}, \mathcal{E}] = \epsilon$.

Our goal is to construct $\mathcal{B}$ that uses $\mathcal{A}$ to win the semantic security game. The reduction algorithm $\mathcal{B}$ interacts with the semantic security challenger and the (message recovery) adversary $\mathcal{A}$. Note that **we will not assume anything about** $\mathcal{A}$ other than the fact that it wins the message recovery game with probability $1/|\mathcal{M}| + \epsilon$.

First attempt: Consider the following algorithm $\mathcal{B}$, and for simplicity, let us assume the message space for security parameter λ is $\{0,1\}^\lambda$. For security parameter λ, the reduction algorithm $\mathcal{B}$ does the following:

1. It first sends $m_0 = 0^\lambda$, $m_1 = 1^\lambda$ to the semantic security challenger.
2. It receives a ciphertext ct, which it sends to the adversary $\mathcal{A}$.
3. The adversary $\mathcal{A}$ sends a message m in response. At this point, a natural thing to do is the following: if $m = 0^\lambda$, $\mathcal{B}$ sends '0', if $m = 1^\lambda$, it sends '1', else it sends a random guess.

Unfortunately, this attempt does not work. Consider the following adversary that wins the message recovery game with non-negligible probability: given a ciphertext, it recovers the underlying message m, and if the message is equal to 0^λ or 1^λ, it sends $\perp$, else it sends m. Note that for this adversary, its winning probability in the message recovery game is close to 1 (in particular, the probability is $1 - 2/2^\lambda$). However, note that this adversary $\mathcal{A}$ is useless for our reduction algorithm $\mathcal{B}$!

Proof. Let $\mathcal{A}$ be a p.p.t. adversary that wins the message recovery game w.r.t. $\mathcal{E}$ with probability $1/|\mathcal{M}| + \epsilon$. We will use $\mathcal{A}$ to build a p.p.t. algorithm $\mathcal{B}$ that wins the semantic security game with advantage ϵ.

We will modify the 'first attempt' reduction algorithm as follows (and we will not assume anything about the message space other than the fact that it's possible to sample uniformly random messages from the message space).

1. The reduction algorithm $\mathcal{B}$ samples two messages m_0, m_1 **uniformly at random** from $\mathcal{M}$. It sends these messages to the semantic security challenger.
2. It receives a ciphertext ct, which it sends to the adversary $\mathcal{A}$.
3. The adversary $\mathcal{A}$ sends a message m in response. If $m = m_0$, $\mathcal{B}$ sends '0', else it sends '1'.

Having defined the reduction algorithm, it is easy to check that $\mathcal{B}$ is probabilistic polynomial time if $\mathcal{A}$ is. We need to show that this algorithm breaks semantic security (with good probability) if $\mathcal{A}$ wins the message recovery game with good probability. We will compute the advantage of $\mathcal{B}$ in the semantic security game (that is, $\mathsf{SSAdv}[\mathcal{B}, \mathcal{E}]$). Recall,

$$\begin{aligned}&\mathsf{SSAdv}[\mathcal{B}, \mathcal{E}](\lambda)\\ =&|\mathsf{Pr}[\mathcal{B} \text{ outputs } 0 \mid \text{Challenger chooses } 0] - \mathsf{Pr}[\mathcal{B} \text{ outputs } 0 \mid \text{Challenger chooses } 1]|\end{aligned}$$

We will analyse each of these probabilities separately.

$$\begin{aligned}&\mathsf{Pr}[\mathcal{B} \text{ outputs } 0 \mid \text{Challenger chooses } 0]\\ =&\mathsf{Pr}\left[\begin{array}{c}\mathcal{B} \text{ chooses } m_0, m_1 \text{ and sends to challenger}\\ \text{Challenger chooses } k \leftarrow \mathcal{K}, \text{ computes ct} \leftarrow \mathsf{Enc}(k, m_0)\\ \text{On receiving ct}, \mathcal{A} \text{ sends } m_0\end{array}\right]\end{aligned}$$

Now, note that this probability is equal to the probability that $\mathcal{A}$ wins the message recovery game, which we assume is $1/|\mathcal{M}| + \epsilon$. This is because $\mathcal{A}$ receives the encryption of a random message and we are interested in the probability that it outputs the random message at the end.

$$\begin{aligned}&\mathsf{Pr}[\mathcal{B} \text{ outputs } 0 \mid \text{Challenger chooses } 1]\\ =&\mathsf{Pr}\left[\begin{array}{c}\mathcal{B} \text{ chooses } m_0, m_1 \text{ and sends to challenger}\\ \text{Challenger chooses } k \leftarrow \mathcal{K}, \text{ computes ct} \leftarrow \mathsf{Enc}(k, m_1)\\ \text{On receiving ct}, \mathcal{A} \text{ sends } m_0\end{array}\right]\\ =&\mathsf{Pr}_{\substack{m_0, m_1, k,\\ r_{enc}, r_{adv}}}[m_0 = \mathcal{A}(\mathsf{Enc}(k, m_1 \,;\, r_{enc}) \,;\, r_{adv})]\\ =&\frac{1}{|\mathcal{M}|}\end{aligned}$$

In the above calculations, the first and second equality simply follow from the descriptions of $\mathcal{A}, \mathcal{B}$ and the semantic-security challenger. The final equality follows from the fact that the choice of m_0 is independent of $\mathcal{A}$'s response.

From the above calculations, it follows that $\mathsf{SSAdv}[\mathcal{B}, \mathcal{E}] = \epsilon$. □

Exercise 3.1. *In the above proof, it was important to choose m_0, m_1 uniformly at random. If just m_0 is chosen at random, but m_1 is some fixed message in the message space, then the argument above does not work. Let us call this reduction $\mathcal{B}_1$ (it picks one of the challenge messages at random, but sets m_1 as a fixed message in the message space).*
Describe an adversary $\mathcal{A}$ that wins the message recovery game with probability $1/|\mathcal{M}| + \epsilon$ where ϵ is non-negligible, but $\mathcal{B}_1$'s advantage in the semantic security game is negligible.

Exercise 3.2. *Consider the following reduction algorithm $\mathcal{B}_2$ (which was proposed by one of the students in class):*

1. *Picks two uniformly random messages $m_0, m_1 \leftarrow \mathcal{M}$ and sends them to the challenger.*
2. *It forwards the challenger's response to the adversary $\mathcal{A}$.*
3. *If the adversary sends m_0, the reduction sends '0', if the adversary sends m_1, then the reduction sends '1', else it sends a uniformly random bit as its guess.*

Suppose the adversary $\mathcal{A}$ wins the message recovery game with probability $1/|\mathcal{M}| + \epsilon$, compute the winning probability of $\mathcal{B}_2$.

Exercise 3.3. *In the reduction described above, the reduction algorithm $\mathcal{B}$ picked two uniformly random messages m_0, m_1 and sends them to the challenger. This is slightly counter-intuitive, since it allows both messages to be the same (in which case the reduction algorithm can surely not succeed in guessing the bit chosen by the challenger). Instead, consider the following reduction algorithm $\mathcal{B}_3$:*

1. *Picks $m_0 \leftarrow \mathcal{M}$, picks $m_1 \leftarrow \mathcal{M} \setminus \{ m_0 \}$, and sends them to the challenger.*
2. *It forwards the challenger's response to the adversary $\mathcal{A}$.*
3. *If the adversary sends m_0, the reduction sends '0', else it sends '1'.*

Suppose the adversary $\mathcal{A}$ wins the message recovery game with probability $1/|\mathcal{M}| + \epsilon$, compute the winning probability of $\mathcal{B}_3$.

The other direction does not hold — there exist encryption schemes that satisfy Definition 3.3 but do not satisfy Definition 3.2. Consider the following encryption scheme with message space $\mathcal{M} = \{0,1\}^{2\lambda}$ and key space $\mathcal{K} = \{0,1\}^{\lambda}$.

- $\mathsf{Enc}(k, m = (m_1, m_2))$: Output $(m_1 \oplus k, m_2)$.
- $\mathsf{Dec}(k, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2))$: Output $(\mathsf{ct}_1 \oplus k, \mathsf{ct}_2)$.

Clearly, this encryption scheme does not satisfy semantic security. Fix any adversary $\mathcal{A}$ (even unbounded), it cannot recover the first half of the message (with non-negligible probability).

3.3 *Our First Cryptographic Building Block: Pseudorandom Generators (a.k.a. Stream Ciphers)*

Lecture 04: August 4th, 2023

The need for cryptographic building blocks/assumptions

In Section 2, we proved that Shannon's one-time pad satisfies perfect semantic security. It would be great if we could construct an encryption scheme (with key space smaller than the message space), and prove that it satisfies Definition 3.2. Unfortunately, this seems very challenging, as we would end up resolving the P vs NP conjecture! As a result, we will give security proofs under widely believed computational assumptions.

Pseudorandom Generators/Stream Ciphers

Our first computational assumption will be the existence of certain length-expanding functions whose output, on a random input, looks uniformly random. Such function families are called *pseudorandom generators*(PRGs). In practice, they are also referred to as stream ciphers. Let us formally define what it means for the output of a function to 'look' uniformly random. We will do this using a security game between a challenger and an adversary.

PRG Game

Intuition: given the function output on a random input, the output looks random

1. Challenger picks $b \leftarrow \{0,1\}$. If $b = 0$, it chooses $s \leftarrow \{0,1\}^{\ell_1(n)}$, sets $u_0 = G_n(s)$. If $b = 1$, it chooses $u_1 \leftarrow \{0,1\}^{\ell_2(n)}$. It sends u_b to $\mathcal{A}$.
2. Adversary outputs its guess b', and wins the game if $b = b'$.

Figure 4: The security game for defining pseudorandom generators is defined with respect to a function family $\mathcal{G}$ and adversary $\mathcal{A}$. The function G_n takes $\ell_1(n)$ bits as input, and outputs $\ell_2(n)$ bits as output.

Definition 3.5. *Let $\ell_1, \ell_2 : \mathbb{N} \to \mathbb{N}$ be polynomials satisfying $\ell_2(i) > \ell_1(i)$ for all $i \in \mathbb{N}$. A collection of deterministic, efficiently-implementable functions*

$$\mathcal{G} = \{ G_n : \{0,1\}^{\ell_1(n)} \to \{0,1\}^{\ell_2(n)} \}_{n \in \mathbb{N}}$$

is a pseudorandom generator if, for any p.p.t. adversary $\mathcal{A}$,

$$\left| \Pr[\mathcal{A} \text{ wins the PRG Game w.r.t. } \mathcal{G}] - \frac{1}{2} \right|$$

is a negligible function. The advantage of $\mathcal{A}$ with respect to $\mathcal{G}$ is defined as

$$\mathsf{PRGAdv}[\mathcal{A}, \mathcal{G}] = \left| \begin{array}{c} \Pr[\mathcal{A} \text{ outputs } 0 \mid \text{Challenger chooses '0'}] \\ -\Pr[\mathcal{A} \text{ outputs } 0 \mid \text{Challenger chooses '1'}] \end{array} \right|$$

◊

Note that the description of $\mathcal{G}$ is public and deterministic, and the adversary can run $G_n(x)$ for any input x. However, the hope is that the evaluation on a random s is indistinguishable from a uniformly random string in the output space.

A PRG is a One-Way-Function

A one-way function family is a family of functions that can be efficiently computed in one direction, but are hard to compute in the other direction. A formal definition is given below.

Definition 3.6. *A collection of efficiently-implementable functions $\mathcal{F} = \{ f_n : \mathcal{X}_n \to \mathcal{Y}_n \}_{n \in \mathbb{N}}$ is a one-way function family if for any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{Pr}_x \left[x' \leftarrow \mathcal{A}(f_n(x)) \wedge f_n(x') = f(x) \right]$$

is a negligible function. ◊

It is easy to show that if a function family $\mathcal{G}$ is a pseudorandom generator *with sufficient length-expansion*, then it is also a one-way function family. A formal statement and proof is included below.

Claim 3.7. *Let $\mathcal{G} = \{ G_n : \{0,1\}^n \to \{0,1\}^{2n} \}_{n \in \mathbb{N}}$. If there exists a p.p.t. adversary $\mathcal{A}$ such that $\mathsf{Pr}[\mathcal{A}$ wins the OWF game w.r.t. $\mathcal{G}]$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}$ such that $\mathsf{PRGAdv}[\mathcal{B}, \mathcal{G}]$ is non-negligible.*

Proof. For ease of notation, we will skip the subscript denoting the security parameter.

Suppose $\mathcal{A}$ wins the OWF game with probability ϵ. Our reduction algorithm $\mathcal{B}$ receives $y \in \{0,1\}^{2n}$, which it forwards to $\mathcal{A}$. The adversary $\mathcal{A}$ sends its response x. If $G(x) = y$, the reduction algorithm guesses '0' (indicating that y is pseudorandom), else it guesses that y is truly random.

To analyse the advantage of $\mathcal{B}$, it suffices to compute the probability of $\mathcal{B}$ outputting '0' conditioned on the challenger picking $b = 0$, and the probability of $\mathcal{B}$ outputting '0' conditioned on the challenger picking $b = 1$.

First, note that $\mathsf{Pr}[\mathcal{B}$ outputs $0 \mid$ Challenger picks '0'$]$ is equal to the adversary's success probability in the one-wayness security game. Hence, this probability is equal to ϵ.

Next, let us compute $\mathsf{Pr}[\mathcal{B}$ outputs $0 \mid$ Challenger picks '1'$]$. In this case, the challenger picks a uniformly random string, and we are interested in the probabilituy that $\mathcal{A}$ finds a 'preimage' for a uniformly random $2n$ bit string. The crucial observation here is that most $2n$ bit strings do not have a pre-image! Since G_n is deterministic, at most 2^n strings have a pre-image, and this is a very small (negligible) fraction of the output space. Let $S = \{ y : y = G_n(x)$ for some $x \in \{0,1\}^n \}$. Note that $|S| \leq 2^n$. Let $\mathsf{BadEvent}$ denote the event that the uniformly random string picked by the challenger is in S. Then $\mathsf{Pr}[\mathsf{BadEvent}] \leq 1/2^n$.

$$
\begin{aligned}
&\Pr[\mathcal{B} \text{ outputs } 0 \mid \text{Challenger picks '1'}] \\
=&\Pr[\mathcal{B} \text{ outputs } 0 \wedge \mathsf{BadEvent} \mid \text{Challenger picks '1'}] \\
&+ \Pr[\mathcal{B} \text{ outputs } 0 \wedge \neg\mathsf{BadEvent} \mid \text{Challenger picks '1'}] \\
\leq&\Pr[\mathsf{BadEvent}] + 0 \\
\leq&\frac{1}{2^n} + 0
\end{aligned}
$$

Note that $\mathcal{B}$ never outputs 0 if BadEvent does not happen, since the $2n$ bit string picked by the challenger does not have a preimage.

To conclude, the reduction algorithm $\mathcal{B}$ has advantage $|\epsilon - 1/2^n|$, which is non-negligible if ϵ is non-negligible. This concludes our proof. □

Exercise 3.4. *Note that in this proof, it was important that $\mathcal{G}$ is a length-doubling function family. A natural question is whether the same result would hold if $\mathcal{G}$ is only mildly expanding; say $G_n : \{0,1\}^n \to \{0,1\}^{n+1}$. The above reduction/analysis would not work, but is the claim true? Either give a proof that works for any length-expanding PRG, or show a counterexample.*

Exponential time/space attacks on PRGs

The following are two simple generic attacks on PRGs. For concreteness, let $G : \{0,1\}^n \to \{0,1\}^{2n}$ (we will skip the 'family of functions' and the 'security parameter subscript' when it is clear from the context).

- Exponential time attack: Given $u \in \{0,1\}^{2n}$, we can iterate over all inputs $x \in \{0,1\}^n$, and check if $u = G(x)$ for some x.
- Exponential space attack: Since the description of G is public, one can precompute a $O(n \cdot 2^n)$ sized (sorted) database, containing $G(x)$ for all $x \in \{0,1\}^n$. Later, when the adversary receives $u \in \{0,1\}^{2n}$, it can perform a binary search in $\mathrm{poly}(n)$ time to check if u is PRG output.

Challenge Problem 2. *Given the description of G, is it possible to prepare a database of size $O\left(2^{n/2}\right)$ s.t., when the adversary receives $u \in \{0,1\}^{2n}$, it can check if u is pseudorandom or not, in $O\left(2^{n/2} \cdot \mathrm{poly}(n)\right)$ steps.*

Pseudorandom generators, in theory and practice

We introduced PRGs as a building block, but a natural question that arises is the following: how are pseudorandom generators built?

In practice, we have very efficient PRGs. The AES circuit can be used as a pseudorandom generator (we will discuss this in Section 4). Another PRG that was widely used in practice is RC4. Today, there are several attacks known against RC4, and therefore it should not be used in practice.

In theoretical cryptography, the initial constructions of PRGs were based on various, well-studied computational assumptions. A remarkable result by Hastad et al. [HILL99] showed that PRGs can be built from any one-way function (establishing that the existence of OWFs is equivalent to the existence of PRGs).

Semantically secure encryption from PRGs

In this section, we will see how to use PRGs for building semantically secure encryption schemes with succinct keys.

Construction 3.8 (Semantically secure encryption from PRGs)**.** *The key space (for security parameter n) is $\mathcal{K}_n = \{0,1\}^n$, and the message space is $\mathcal{M}_n = \{0,1\}^{\ell(n)}$, where ℓ is some polynomial.*
Let $\mathcal{G} = \{ G_n : \{0,1\}^n \to \{0,1\}^{\ell(n)} \}_{n\in\mathbb{N}}$ be a secure PRG family. We will skip the dependence on security parameter when it is clear from the context.

The encryption scheme $\mathcal{E}_\mathcal{G} = (\mathsf{Enc}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Enc}(k, m) = m \oplus G(k)$
- $\mathsf{Dec}(k, \mathsf{ct}) = \mathsf{ct} \oplus G(k)$

$\Diamond$

Claim 3.9. *Suppose there exists a prob. poly. time adversary $\mathcal{A}$ and a non-negligible function ϵ such that $\mathsf{SSAdv}[\mathcal{A}, \mathcal{E}_\mathcal{G}] = \epsilon$. Then there exists a prob. poly. time algorithm $\mathcal{B}$ and a non-negligible function $\epsilon_\mathcal{B}$ such that $\mathsf{PRGAdv}[\mathcal{B}, \mathcal{G}] = \epsilon_\mathcal{B}$.*

It is not immediately clear how to use $\mathcal{A}$ to break the PRG security. This is because $\mathcal{A}$ can distinguish between $G(k) \oplus m_0$ and $G(k) \oplus m_1$ (for a random k), but the reduction algorithm must distinguish between $G(k)$ and a uniformly random string. The key idea here is to define intermediate 'hypothetical experiments'.

Let *World 0* denote the scenario where the adversary interacts with a challenger. It sends two messages m_0, m_1, the challenger encrypts m_0 using a uniformly random key (in this case, the challenger sends $G(k) \oplus m_0$), and the adversary finally outputs a bit b'. Similarly, let *World 1* denote the scenario where the challenger sends an encryption of m_1 (that is, $G(k) \oplus m_1$). Since $\mathcal{A}$ breaks the security of $\mathcal{E}_\mathcal{G}$, the probability of outputting 0 in World-0 (let's denote this probability by p_0) is far-apart from the probability of outputting 0 in World-1 (p_1). Consider two intermediate worlds (*Hybrid-World-0* and *Hybrid-World-1*) where the challenger sends $\mathsf{random}_0 \oplus m_0$ and $\mathsf{random}_1 \oplus m_1$ respectively. Let $p_{\mathsf{Hyb},0}$ and $p_{\mathsf{Hyb},1}$ denote the probability of adversary outputting 0 at the end of the experiment in Hybrid-World-0 and Hybrid-World-1 respectively. First, note that using security of Shannon's one-time-pad, these two probabilities must be equal.

Next, note that since p_0 and p_1 are far-apart (and $p_{\mathsf{Hyb},0} = p_{\mathsf{Hyb},1}$), either p_0 and $p_{\mathsf{Hyb},0}$ are far-apart, or p_1 and $p_{\mathsf{Hyb},1}$ are far-apart. In both these cases, we can use $\mathcal{A}$ to break the PRG security.

Lecture 5: August 8th, 2023

Proof. Let p_0 (resp. p_1) denote the probability of $\mathcal{A}$ outputting 0 in world-0 (resp. world-1), where world-0 and world-1 are defined below.

World 0:

1. $\mathcal{A}$ sends m_0, m_1.
2. Chall. chooses $k \leftarrow \{0,1\}^n$, computes $r = G(k)$.
   It sends $m_0 \oplus r$.
3. $\mathcal{A}$ outputs b'.

$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_0$

World 1:

1. $\mathcal{A}$ sends m_0, m_1.
2. Chall. chooses $k \leftarrow \{0,1\}^n$, computes $r = G(k)$.
   It sends $m_1 \oplus r$.
3. $\mathcal{A}$ outputs b'.

$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_1$

Next, we will define two hybrid worlds: hybrid-world-0, and hybrid-world-1. Hybrid-world-0 (resp. hybrid-world-1) is exactly identical to world-0 (resp. world-1), except that r is chosen uniformly at random (instead of being computed using key k).

Hybrid-World-0:

1. $\mathcal{A}$ sends m_0, m_1.
2. Chall. chooses $r \leftarrow \{0,1\}^{\ell(n)}$. It sends $m_0 \oplus r$.
3. $\mathcal{A}$ outputs b'.

$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},0}$

Hybrid-World-1:

1. $\mathcal{A}$ sends m_0, m_1.
2. Chall. chooses $r \leftarrow \{0,1\}^{\ell(n)}$. It sends $m_1 \oplus r$.
3. $\mathcal{A}$ outputs b'.

$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},1}$

First, note that $p_{\mathsf{Hyb},0} = p_{\mathsf{Hyb},1}$. This follows from the perfect secrecy of Shannon's One Time Pad.

Since $|p_0 - p_1| = \epsilon$, it follows that either $\left|p_0 - p_{\mathsf{Hyb},0}\right| \geq \epsilon/2$, or $\left|p_1 - p_{\mathsf{Hyb},1}\right| \geq \epsilon/2$. In both these cases, we have a reduction algorithm that breaks the PRG security with probability at least $1/2 + \epsilon/4$.

Observation 3.10. *Let $\mathcal{A}$ be a p.p.t. adversary, and let p_0 and $p_{\mathsf{Hyb},0}$ denote the probability of $\mathcal{A}$ outputting 0 in World-0 and Hybrid-World-0 respectively. Then there exists a p.p.t. algorithm $\mathcal{B}_0$ such that* $\mathsf{PRGAdv}[\mathcal{B}_0, \mathcal{G}] = \left|p_0 - p_{\mathsf{Hyb},0}\right|$.

Proof. The reduction algorithm $\mathcal{B}_0$ works as follows. It receives m_0, m_1 from $\mathcal{A}$ and a string $y \in \{0,1\}^{\ell(n)}$ from the PRG challenger. It sends $m_0 \oplus y$ to $\mathcal{A}$, after which $\mathcal{A}$ sends a bit b'. If $b' = 0$, the reduction algorithm sends 0 to the challenger (indicating that y is pseudorandom), else it sends 1.

Let us compute the advantage of $\mathcal{B}_0$.

$$\begin{aligned}
&\mathsf{Pr}[\mathcal{B}_0 \text{ sends } 0 \mid \text{Challenger chooses } 0] \\
=&\mathsf{Pr}[\mathcal{A} \text{ sends } 0 \text{ in World-0}] \\
=&p_0
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Pr}[\mathcal{B}_0 \text{ sends } 0 \mid \text{Challenger chooses } 1] \\
=&\mathsf{Pr}[\mathcal{A} \text{ sends } 0 \text{ in Hybrid-World-0}] \\
=&p_{\mathsf{Hyb},0}
\end{aligned}$$

From here, it follows that $\mathsf{PRGAdv}[\mathcal{B}_0, \mathcal{G}] = \left|p_0 - p_{\mathsf{Hyb},0}\right|$. □

Observation 3.11. *Let $\mathcal{A}$ be a p.p.t. adversary, and let p_1 and $p_{\mathsf{Hyb},1}$ denote the probability of $\mathcal{A}$ outputting 0 in World-1 and Hybrid-World-1 respectively. Then there exists a p.p.t. algorithm $\mathcal{B}_1$ such that* $\mathsf{PRGAdv}[\mathcal{B}_1, \mathcal{G}] = \left|p_{\mathsf{Hyb},1} - p_1\right|$.

Proof. The proof of this observation is very similar to the proof of Observation 3.10. However, we will make a small modification (which is color-coded) that will be useful for puting $\mathcal{B}_0$ and $\mathcal{B}_1$ together at the end of our proof.

The reduction algorithm $\mathcal{B}_1$ receives m_0, m_1 from $\mathcal{A}$ and a string $y \in \{0,1\}^{2n}$ from the PRG challenger. It sends $m_1 \oplus y$ to $\mathcal{A}$, after which $\mathcal{A}$ sends a bit b'. If $b' = 0$, the reduction algorithm sends 1 to the challenger (indicating that y is random), else it sends 0.

Note that we could have considered a different reduction $\mathcal{B}_1'$ that simply forwards the response of $\mathcal{A}$. While this reduction would also suffice for the proof of Observation 3.11, we will not be able to define the 'combined' reduction $\mathcal{B}$ (which is defined after the proof of Observation 3.11).

Let us compute the advantage of $\mathcal{B}_1$.

$$\begin{aligned}
&\mathsf{Pr}\left[\mathcal{B}_1 \text{ sends } 0 \mid \text{Challenger chooses } 0\right] \\
=&\mathsf{Pr}\left[\mathcal{A} \text{ sends } 1 \text{ in World-1}\right] \\
=&1 - \mathsf{Pr}\left[\mathcal{A} \text{ sends } 0 \text{ in World-1}\right] \\
=&1 - p_1
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Pr}\left[\mathcal{B}_1 \text{ sends } 0 \mid \text{Challenger chooses } 1\right] \\
=&\mathsf{Pr}\left[\mathcal{A} \text{ sends } 1 \text{ in Hybrid-World-1}\right] \\
=&1 - \mathsf{Pr}\left[\mathcal{A} \text{ sends } 0 \text{ in Hybrid-World-1}\right] \\
=&1 - p_{\mathsf{Hyb},1}
\end{aligned}$$

From here, it follows that $\mathsf{PRGAdv}[\mathcal{B}_1, \mathcal{G}] = \left|p_1 - p_{\mathsf{Hyb},1}\right|$. □

Putting everything together: recall, our objective was to construct a reduction algorithm $\mathcal{B}$ such that it breaks PRG security with non-negligible advantage (assuming $\mathcal{A}$ breaks semantic security of $\mathcal{E}_{\mathcal{G}}$). As a result, we need to combine the two reductions $\mathcal{B}_0$ and $\mathcal{B}_1$. We know that either $\mathcal{B}_0$ or $\mathcal{B}_1$ has non-negligible advantage. A natural idea : pick one of them uniformly at random. That is, the reduction algorithm $\mathcal{B}$ picks $\beta \leftarrow \{0,1\}$, and runs $\mathcal{B}_\beta$. Using this approach, we get that

$$\begin{aligned}
&\mathsf{Pr}\left[\mathcal{B} \text{ sends } 0 \mid \text{Challenger chooses } 0\right] \\
=&\frac{1}{2}p_0 + \frac{1}{2}\left(1 - p_1\right)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Pr}\left[\mathcal{B} \text{ sends } 0 \mid \text{Challenger chooses } 1\right] \\
=&\frac{1}{2}p_{\mathsf{Hyb},0} + \frac{1}{2}\left(1 - p_{\mathsf{Hyb},1}\right)
\end{aligned}$$

Hence, the advantage of $\mathcal{B}$ in the PRG security game w.r.t. $\mathcal{G}$ is

$$\mathsf{PRGAdv}[\mathcal{B}, \mathcal{G}] = \frac{1}{2}|p_0 - p_1|.$$

□

*The **hybrid proof technique** is an important proof technique in modern cryptography. Most of the creativity in theoretical crypto research goes into designing appropriate hybrid-worlds. It is important to design these hybrid experiments in such a manner that the indistinguishability of consecutive hybrids relies on only one cryptographic building block. We will see several examples of the hybrid proof technique in this course.*

In the above proof, we defined a 'combined reduction' at the end of the proof. For this course, it suffices to just define the hybrids appropriately, and give a reduction for the consecutive hybrids.

Expanding PRG output length

The PRGs used in practice are *stream ciphers*, which means that they can produce as many pseudorandom bits as needed. However, there are PRG constructions where the output length is some fixed polynomial in the input length (this is especially the case in many theoretical constructions; we will see one of them in Section 3.3). For such PRG constructions, it is natural to ask whether it is possible to increase the output length. Besides being a natural question, it will be yet another example of the hybrid proof technique, and also illustrates some incorrect uses of PRGs.

Let $\mathcal{G} = \{ G_n : \{0,1\}^n \to \{0,1\}^{2n} \}_{n\in\mathbb{N}}$ be a secure PRG family. We want to build a PRG family with greater 'stretch' — that is, a function family $\mathcal{G}' = \{ G'_n \}_{n\in\mathbb{N}}$ where G'_n maps n bits to $\ell \cdot n$ bits, for some $\ell > 2$. For simplicity, let us start with $\ell = 4$.

Of course, there are a number of ways of achieving this, but how to ensure that the resulting function family is also a secure PRG (assuming $\mathcal{G}$ is a secure PRG)? The following candidates were proposed in class.

Construction 3.12 (PRG output expansion: Candidate 1)**.**

$$G' : \{0,1\}^n \to \{0,1\}^{4n}$$
$$G'(s) = (s_0, s_1, s_2, s_3) \text{ where}$$
$$(s'_0, s'_1) = G(s), (s_0, s_1) = G(s'_0), (s_2, s_3) = G(s'_1)$$

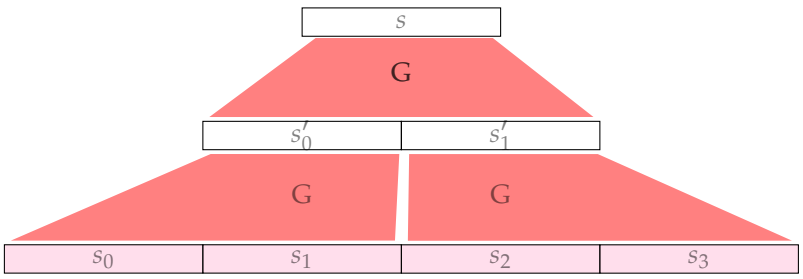


Figure 5: PRG length expansion via parallel application of G. The colored blocks are the output of G'.

This is a secure PRG, and we will see the proof of security later in this section.

◊

Construction 3.13 (PRG output expansion: Candidate 2)**.**

$$G' : \{0,1\}^n \to \{0,1\}^{4n}$$
$$G'(s) = (s_0, s_1, s_2, s_3) \text{ where}$$
$$(s_0, s_1) = G(s), (s_2, s_3) = G(s_1)$$

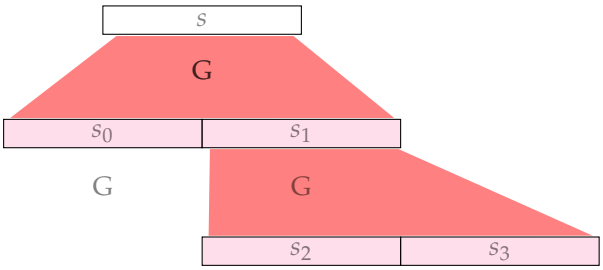


Figure 6: The colored blocks are the output of G'.

This candidate is not a secure PRG. Given a $4n$ *bit string* (y_0, y_1, y_2, y_3)*, the adversary can check if* $(y_2, y_3) = G(y_1)$*. If so, it guesses that this is a pseudo-random string, else it guesses that it is random. Check that this adversary wins the PRG game with probability close to* 1. ◊

Construction 3.14 (PRG output expansion: Candidate 3)**.**

$$G' : \{0,1\}^n \to \{0,1\}^{4n}$$
$$G'(s) = G(s) \,||\, G(\bar{s})$$

In this construction, the first $2n$ *bits are obtained by applying* G *on* s*, and the remaining* $2n$ *bits are obtained by applying* G *on* $\bar{s} = s \oplus 1^n$*. While this construction is not immediately broken, the security of this candidate does not follow from the PRG security of* G*. In fact, similar candidates were used in practice, and practical attacks were demonstrated against these candidates.* ◊

Construction 3.15 (PRG output expansion: Candidate 4)**.**

$$G' : \{0,1\}^n \to \{0,1\}^{4n}$$
$$G'(s) = (s_0, s_1, s_2, s_3) \text{ where}$$
$$(s'_0, s'_1) = G(s), (s_0, s_1) = G(s'_0), (s_2, s_3) = G(s'_0[n/2+1, n] \,||\, s'_1[1, n/2])$$

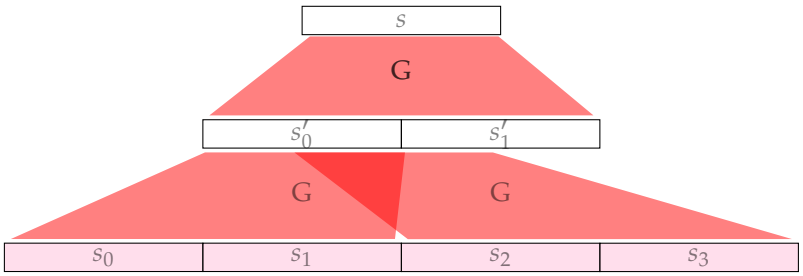


Figure 7: PRG length expansion via parallel application of G.

This construction is similar to Construction 3.12. The first $2n$ bits of the output are derived from s'_0, while the remaining $2n$ bits are derived by applying G to $s'_0[n/2+1, n] \,||\, s'_1[1, n/2]$ (that is, the concatention of the right half of s'_0 and the left half of s'_1). This construction is not immediately broken. However, its security does not follow from the PRG security of G. There exist secure PRGs which, when composed in this manner, result in an insecure construction. ◊

Construction 3.16 (PRG output expansion: Candidate 5)**.**

$$G' : \{0,1\}^n \to \{0,1\}^{4n}$$
$$G'(s) = (s_0, s_1, s_2, s_3) \text{ where}$$
$$(s_0, s'_1) = G(s), (s_1, s'_2) = G(s'_1), (s_2, s_3) = G(s'_2)$$

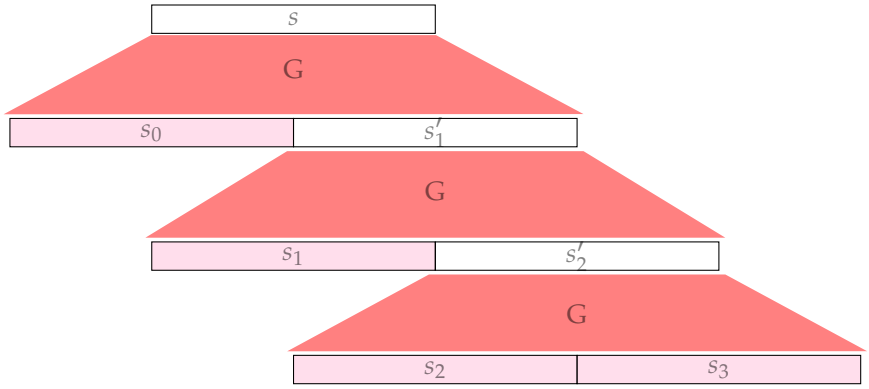


Figure 8: PRG length expansion via sequential application of G.

The above construction is provably secure; it was given by Manuel Blum and Silvio Micali, two very famous names in cryptography (and both got the Turing award for various contributions to the foundations of cryptography). ◊

Some of the above candidates are secure, others may or may not be secure, and finally, there are some that are broken. Construction 3.12 and Construction 3.16 are secure candidates, and the corresponding security proofs are presented

below. Both proofs involve the 'hybrid proof technique'.

Claim 3.17. *Let G' denote the candidate PRG construction described in Construction 3.12. Suppose there exists a p.p.t. adversary $\mathcal{A}$ such that $\mathsf{PRGAdv}[\mathcal{A}, \mathcal{G}']$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}$ such that $\mathsf{PRGAdv}[\mathcal{B}, \mathcal{G}]$ is non-negligible.*

Proof. Let $p_b = \mathsf{Pr}[\mathcal{A} \text{ outputs } 0 \mid \text{Challenger picks b}]$. We are given that $|p_0 - p_1|$ is non-negligible. Let World-0 denote the setting where the challenger picks 0 (and sends a $4n$ bit pseudorandom string), and World-1 denote the setting where the challenger picks 1 (and sends a uniformly random $4n$ bit string).

These two worlds are described below.

World 0:

1. Chall. chooses $s \leftarrow \{0,1\}^n$, computes
   - $(s'_0, s'_1) = G(s)$
   - $(s_0, s_1) = G(s'_0)$
   - $(s_2, s_3) = G(s'_1)$

   It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_0$$

World 1:

1. Chall. chooses $s_0, s_1, s_2, s_3 \leftarrow \{0,1\}^n$. It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_1$$

Next, we will define two hybrid worlds: hybrid-world-0, and hybrid-world-1.

Hybrid-World-0:

1. Chall. chooses $s'_0, s'_1 \leftarrow \{0,1\}^n$,
   - $(s_0, s_1) = G(s'_0)$
   - $(s_2, s_3) = G(s'_1)$

   It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},0}$$

Hybrid-World-1:

1. Chall. chooses $s_0, s_1, s'_1 \leftarrow \{0,1\}^n$,
   - $(s_2, s_3) = G(s'_1)$

   It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},1}$$

Lecture 06: August 11th, 2023

Observation 3.18. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $|p_0 - p_{\mathsf{Hyb},0}| = \epsilon$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}_1$ such that $\mathsf{PRGAdv}[\mathcal{B}_1, G] = \epsilon$.*

Proof. The reduction algorithm receives (y_0, y_1) from the PRG challenger. It computes $(s_0, s_1) = G(y_0)$, $(s_2, s_3) = G(s_1)$ and sends (s_0, s_1, s_2, s_3) to $\mathcal{A}$. It forwards the guess of $\mathcal{A}$ to the PRG challenger.

If (y_0, y_1) are pseudorandom, then this corresponds to World-0, else it corresponds to Hybrid-World-0. Check that $\mathsf{PRGAdv}[\mathcal{B}_1, G] = |p_0 - p_{\mathsf{Hyb},0}|$. □

Observation 3.19. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $|p_{\mathsf{Hyb},0} - p_{\mathsf{Hyb},1}| = \epsilon$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}_2$ such that $\mathsf{PRGAdv}[\mathcal{B}_2, G] = \epsilon$.*

Proof. The reduction algorithm receives (y_0, y_1) from the PRG challenger. It chooses s'_1, computes $(s_2, s_3) = G(s'_1)$ and sends (y_0, y_1, s_2, s_3) to $\mathcal{A}$. It forwards the guess of $\mathcal{A}$ to the PRG challenger.

Note that if the reduction sends (s_2, s_3, y_0, y_1) to the adversary, then this proof does not work (why?). For similar reasons, the reduction should not send (y_1, y_0, s_2, s_3).

If (y_0, y_1) are pseudorandom, then this corresponds to Hybrid-World-0, else it corresponds to Hybrid-World-1.Check that $\mathsf{PRGAdv}[\mathcal{B}_2, G] = \left|p_{\mathsf{Hyb},0} - p_{\mathsf{Hyb},1}\right|$. □

Observation 3.20. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $\left|p_{\mathsf{Hyb},1} - p_1\right| = \epsilon$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}_3$ such that $\mathsf{PRGAdv}[\mathcal{B}_3, G] = \epsilon$.*

Proof. The reduction algorithm receives (y_0, y_1) from the PRG challenger. It chooses s_0, s_1, and sends (s_0, s_1, y_0, y_1) to $\mathcal{A}$. It forwards the guess of $\mathcal{A}$ to the PRG challenger.

If (y_0, y_1) are pseudorandom, then this corresponds to Hybrid-World-1, else it corresponds to World-1.Check that $\mathsf{PRGAdv}[\mathcal{B}_3, G] = \left|p_{\mathsf{Hyb},1} - p_1\right|$. □

This concludes our proof. One can also construct a 'combined' reduction $\mathcal{B}$ that picks one of $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ uniformly at random. Check that if $\mathsf{PRGAdv}[\mathcal{A}, G'] = \epsilon$, then $\mathsf{PRGAdv}[\mathcal{B}, G] = \frac{\epsilon}{3}$. □

(*) **Exercise 3.5.** *Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be a semantically secure encryption scheme with key space $\mathcal{K} = \{0,1\}^n$, message space $\mathcal{M} = \{0,1\}^{\ell(n)}$ and ciphertext space $\mathcal{C} = \{0,1\}^{2\ell(n)}$. Consider the following encryption algorithm $\mathcal{E}'$:*

- $\mathsf{Enc}'(k, m)$*: Compute* $\mathsf{ct} \leftarrow \mathsf{Enc}(k, m)$*. Parse* ct *as* $(\mathsf{ct}_1, \mathsf{ct}_2)$ *where* $\mathsf{ct}_i \in \{0,1\}^{\ell(n)}$*. Compute* $\mathsf{ct}'_i \leftarrow \mathsf{Enc}(k, \mathsf{ct}_i)$ *and output* $(\mathsf{ct}'_1, \mathsf{ct}'_2)$ *as the ciphertext.*
- $\mathsf{Dec}'(k, (\mathsf{ct}'_1, \mathsf{ct}'_2))$*: Compute* $\mathsf{ct}_i = \mathsf{Dec}(k, \mathsf{ct}'_i)$*, and output* $\mathsf{Dec}(k, (\mathsf{ct}_1, \mathsf{ct}_2))$.

Show that there exist semantically secure encryption schemes $\mathcal{E}$ such that the above transformation results in an insecure encryption scheme. Check that the hybrid proof technique discussed in the proof of Claim 3.17 does not work for semantically secure encryption schemes.

Next, we will prove the security of construction 3.16.

Claim 3.21. *Let G' denote the candidate PRG construction described in Construction 3.16. Suppose there exists a p.p.t. adversary $\mathcal{A}$ such that $\mathsf{PRGAdv}[\mathcal{A}, \mathcal{G}']$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}$ such that $\mathsf{PRGAdv}[\mathcal{B}, \mathcal{G}]$ is non-negligible.*

Proof. Let $p_b = \mathsf{Pr}[\mathcal{A}$ outputs $0 \mid$ Challenger chooses $b]$, and $|p_0 - p_1| = \epsilon$ which is non-negligible. We will define intermediate hybrid worlds to connect World-0 (where challenger chooses 0) and World-1 (where challenger chooses 1).

World 0:

1. Chall. chooses $s \leftarrow \{0,1\}^n$, computes
   - $(s_0, s_1') = G(s)$
   - $(s_1, s_2') = G(s_1')$
   - $(s_2, s_3) = G(s_2')$

   It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_0$$

World 1:

1. Chall. chooses $s_0, s_1, s_2, s_3 \leftarrow \{0,1\}^n$. It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_1$$

Next, we will define two hybrid worlds: hybrid-world-0, and hybrid-world-1.

Hybrid-World-0:

1. Chall. chooses $s_0, s_1' \leftarrow \{0,1\}^n$,
   - $(s_1, s_2') = G(s_1')$
   - $(s_2, s_3) = G(s_2')$

   It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},0}$$

Hybrid-World-1:

1. Chall. chooses $s_0, s_1, s_2' \leftarrow \{0,1\}^n$,
   - $(s_2, s_3) = G(s_2')$

   It sends (s_0, s_1, s_2, s_3).
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},1}$$

Having defined the hybrids, we will show that the consecutive hybrids are indistinguishable, assuming G is secure (or, taking the contrapositive, if two consecutive hybrids are distinguishable, then there exists an attack on G).

Observation 3.22. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $\left|p_0 - p_{\mathsf{Hyb},0}\right| = \epsilon$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}_1$ such that* $\mathsf{PRGAdv}[\mathcal{B}_1, G] = \epsilon$.

Proof. The reduction algorithm receives $(u_0, u_1) \in \{0,1\}^{2n}$ from the PRG challenger for G. It computes $(s_1, s_2') = G(u_1)$, $(s_2, s_3) = G(s_2')$, and sends (u_0, s_1, s_2, s_3) to $\mathcal{A}$. The adversary sends its guess b'. If $b' = 0$, $\mathcal{B}_1$ guesses that (u_0, u_1) is pseudorandom, else it guesses that it is random.

Please verify that

$$\Pr[\mathcal{B}_1 \text{ sends } 0 \mid \text{Challenger chooses } 0] = p_0$$
$$\Pr[\mathcal{B}_1 \text{ sends } 0 \mid \text{Challenger chooses } 1] = p_{\mathsf{Hyb},0}$$

Therefore, $\mathsf{PRGAdv}[\mathcal{B}_1, G] = \left|p_0 - p_{\mathsf{Hyb},0}\right|$. □

Observation 3.23. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $\left|p_{\mathsf{Hyb},0} - p_{\mathsf{Hyb},1}\right| = \epsilon$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}_2$ such that* $\mathsf{PRGAdv}[\mathcal{B}_2, G] = \epsilon$.

Proof. The reduction algorithm receives $(u_0, u_1) \in \{0,1\}^{2n}$ from the PRG challenger for G. It chooses $s_0 \leftarrow \{0,1\}^n$, computes $(s_2, s_3) = G(u_1)$, and sends (s_0, u_0, s_2, s_3) to $\mathcal{A}$. The adversary sends its guess b'. If $b' = 0$, $\mathcal{B}_2$ guesses that (u_0, u_1) is pseudorandom, else it guesses that it is random.

Please verify that

$$\Pr[\mathcal{B}_2 \text{ sends } 0 \mid \text{Challenger chooses } 0] = p_{\mathsf{Hyb},0}$$
$$\Pr[\mathcal{B}_2 \text{ sends } 0 \mid \text{Challenger chooses } 1] = p_{\mathsf{Hyb},1}$$

Therefore, $\mathsf{PRGAdv}[\mathcal{B}_2, G] = \left|p_{\mathsf{Hyb},0} - p_{\mathsf{Hyb},1}\right|$. □

Observation 3.24. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $\left|p_{\mathsf{Hyb},1} - p_1\right| = \epsilon$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}_3$ such that $\mathsf{PRGAdv}[\mathcal{B}_3, G] = \epsilon$.*

Proof. The reduction algorithm receives $(u_0, u_1) \in \{0,1\}^{2n}$ from the PRG challenger for G. It chooses $s_0, s_1 \leftarrow \{0,1\}^n$, and sends (s_0, s_1, u_0, u_1) to $\mathcal{A}$. The adversary sends its guess b'. If $b' = 0$, $\mathcal{B}_3$ guesses that (u_0, u_1) is pseudorandom, else it guesses that it is random.

Please verify that

$$\Pr[\mathcal{B}_3 \text{ sends } 0 \mid \text{Challenger chooses } 0] = p_{\mathsf{Hyb},1}$$
$$\Pr[\mathcal{B}_3 \text{ sends } 0 \mid \text{Challenger chooses } 1] = p_1$$

Therefore, $\mathsf{PRGAdv}[\mathcal{B}_3, G] = \left|p_{\mathsf{Hyb},1} - p_1\right|$. □

Putting everything together: our overall reduction $\mathcal{B}$ will pick one of the reductions $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ at random. Intuitively, the reduction guesses whether $\left|p_0 - p_{\mathsf{Hyb},0}\right|, \left|p_{\mathsf{Hyb},0} - p_{\mathsf{Hyb},1}\right|$ or $\left|p_{\mathsf{Hyb},1} - p_1\right|$ is non-negligible. Check that the advantage of $\mathcal{B}$ in the PRG game is

$$\mathsf{PRGAdv}[\mathcal{B}, G] = \frac{1}{3}|p_0 - p_1| = \frac{1}{3}\mathsf{PRGAdv}\left[\mathcal{A}, G'\right]$$

which is non-negligible if $\mathsf{PRGAdv}[\mathcal{A}, G']$ is non-negligible. □

(*) Exercise 3.6. *The following hybrid structure was proposed by one of the students for Construction 3.16.*

Hybrid-World-0:

1. *Chall. chooses $s, s_2, s_3 \leftarrow \{0,1\}^n$,*
   - $(s_0, s_1') = G(s)$
   - $(s_1, s_2') = G(s_1')$

   It sends (s_0, s_1, s_2, s_3).
2. *$\mathcal{A}$ outputs b'.*

Hybrid-World-1:

1. *Chall. chooses $s, s_1, s_2, s_3 \leftarrow \{0,1\}^n$,*
   - $(s_0, s_1') = G(s)$

   It sends (s_0, s_1, s_2, s_3).
2. *$\mathcal{A}$ outputs b'.*

Which of the following conclusions are correct:

- *If $\exists$ a p.p.t. adversary that distinguishes between World-0 and Hybrid-World-0, then $\exists$ a p.p.t. reduction that breaks PRG security of G.*
- *If $\exists$ a p.p.t. adversary that distinguishes between Hybrid-World-0 and Hybrid-World-1, then $\exists$ a p.p.t. reduction that breaks PRG security of G.*
- *If $\exists$ a p.p.t. adversary that distinguishes between Hybrid-World-1 and World-1, then $\exists$ a p.p.t. reduction that breaks PRG security of G.*

(*) Exercise 3.7. *Recall Construction 3.13. In this candidate, $G' : \{0,1\}^n \to \{0,1\}^{4n}$ that uses $G : \{0,1\}^n \to \{0,1\}^{2n}$ as a building block:*

$$G'(s) = (s_0, s_1, s_2, s_3) \text{ where}$$
$$(s_0, s_1) = G(s), (s_2, s_3) = G(s_1)$$

First, show that this is not a secure PRG. Next, check where does the above hybrid proof technique (discussed in proof of Claim 3.21) fail for this construction.

A (theoretical) PRG construction

In this section, we present a PRG construction based on a well-studied computational problem called *Learning with Errors*. Let us first see the computational problem. We will state a specific version of this problem, although it can be appropriately generalized.

Computational Problem 1 (Learning with Errors (LWE))**.** *Let $q = \Theta(2^n)$ be an n bit prime, and let $m = n^2$. Consider the following two distributions:*

$$\mathcal{D}_0 = \{ (\mathbf{A}, \mathbf{b}) \ : \ \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow [-n, n]^m, \mathbf{b} = \mathbf{s}^\top \cdot \mathbf{A} + \mathbf{e} \bmod q \}$$
$$\mathcal{D}_1 = \{ (\mathbf{A}, \mathbf{u}) \ : \ \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{u} \leftarrow \mathbb{Z}_q^m \}$$

In words, a sample from $\mathcal{D}_0$ is computed as follows: sample a uniformly random matrix $\mathbf{A}$, a uniformly random vector $\mathbf{s}$, an error vector *$\mathbf{e} \leftarrow [-n, n]^m$, compute $\mathbf{b} = \mathbf{s}^\top \cdot \mathbf{A} + \mathbf{e}$ and output $(\mathbf{A}, \mathbf{b})$. A sample from $\mathcal{D}_1$ consists of a uniformly random vector $\mathbf{A}$ and a uniformly random vector $\mathbf{b}$.*

The Learning with Errors (LWE) Assumption states that no p.p.t. algorithm can distinguish between $\mathcal{D}_0$ and $\mathcal{D}_1$.

A few observations about the LWE problem/assumption:

- if the 'error vector' $\mathbf{e}$ was sampled uniformly at random from $\mathbb{Z}_q^m$, then these two distributions would be identical.
- Without the error vector, one can easily distinguish between these two distributions by using Gaussian elimination. Somewhat surprisingly, when the error vector is added, we don't have any polynomial time algorithm for this problem. Any such algorithm would end up resolving decades old computational problems.

  However, a computationally unbounded adversary **can** distinguish between these two distributions.

  Discuss with me in case you're interested in learning more about why LWE is believed to be hard, or why it is interesting for post-quantum cryptography.
- The LWE problem seems to be resilient against quantum algorithms too! This makes it one of the leading candidates for *post-quantum cryptography* (cryptography that is secure in the presence of quantum adversaries).

Using the LWE problem, we can build a PRG G with appropriate domain and range, such that if there is an attack on the PRG, then there a p.p.t. algorithm for solving LWE. We assume the modulus q (an n bit prime) for security parameter n is fixed and known to everyone.

For security parameter n, the domain and co-domain will be $\ell_1(n)$ and $\ell_2(n)$ where ℓ_1 and ℓ_2 are appropriately chosen polynomials. It is a valid PRG as long as $\ell_2 > \ell_1$.

Construction 3.25 (PRG construction based on LWE)**.** *Let* $\ell_1(n) = n^4 + n^2 + n^2(\log n + 1)$, $\ell_2(n) = n^4 + n^3$. *On input* $\mathbf{x} \in \{0,1\}^{\ell_1(n)}$, *parse the input as follows:*

- *use the first* n^4 *bits to define a matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$.
- *use the next* n^2 *bits to define a vector* $\mathbf{s} \in \mathbb{Z}_q^n$.
- *use the remaining bits to define a vector* $\mathbf{e} \in [-n, n]^m$.

Output $(\mathbf{A}, \mathbf{s}^\top \cdot \mathbf{A} + \mathbf{e} \bmod q)$. ◊

Check that the function is input-expanding. The proof of security follows from the LWE assumption.

Some incorrect uses of PRGs

In this section, we highlight some incorrect uses of PRGs. These are applications taken from real-world attacks.

- The semantically secure encryption scheme derived from PRGs is only one-time secure. The Russian spies, however, ignored the seriousness of the two-time pad attack, and used the same key for multiple encryption. The US intelligence intercepted and decoded many of the 'encrypted' conversations. The details of this effort, known as *Project Venona* are now publicly available.
- Microsoft's PPTP protocol between a server and a client was designed as follows: both the client and the server share a key k, which is used as the PRG input. The client sends $\mathsf{clientmsg} \oplus G(k)$, and the server responds with $\mathsf{servermsg} \oplus G(k)$. This is again an example of a scenario where the two-time-pad attack applies.

  To prevent this attack, the client and server should have shared two different keys k_c and k_s.
- One of the early WiFi protocols needed to generate a random number periodically. It used a PRG G and worked as follows. Let t denote the time-step.

  For $t = 1$ to ∞:

  1. Output $G(s \oplus t)$.

  This stream of bits may not look random. This is because the seeds used at each time-step are all related, leading to *related-key attacks*. Such attacks were practically demonstrated when the RC4 PRG was used for the WiFi protocol.

  Ideally, for each time step, a different seed should have been used. However, sampling such a seed at each time step can be expensive/infeasible for a low-computation device such as the WiFi router. One secure approach would be to use a *pseudorandom function* instead of a PRG, and we will see this primitive in Section 4.

Chapter Summary and References

- We introduced the security parameter, negligible functions and saw how to define computational security.
- Next, we discussed the first security proof in the computational setting - showing that semantic security implies security against message recovery attacks. In the first quiz, we saw that semantic security implies security against key recovery attacks.
- Next, we introduced the first cryptographic building block - pseudorandom generators. Practitioners refer to this object as 'stream ciphers'. We showed that a PRG must also be a one-way function, and then discussed how to use PRGs for semantic security. This proof introduced a useful proof technique called the 'hybrid proof technique'.
- We then saw how to increase the 'stretch' of a PRG. We saw two provably secure ways of doing this. Again, the main takeaway here is the hybrid proof technique. Even if the construction is secure, it is important to define the intermediate hybrids appropriately.
- Finally, we discussed a PRG construction based on the Learning with Errors problem, and some insecure applications of PRG.

Textbook References: Boneh-Shoup (v6, [BS23]), Sections 2.2, 3.1, 3.2, 3.3, 3.4

4 Block Ciphers and Many-Time Security

Last section, we discussed about pseudorandom generators, and saw that it suffices for achieving semantic security for encryption schemes. However, the encryption scheme we saw in the last section is not secure against two-time attacks. In this section, we will introduce a new cryptographic object which is more powerful than PRGs, and will suffice for building encryption schemes secure against all passive adversaries. This cryptographic building block is called a *pseudorandom function*. The most widely used cryptographic algorithm — AES — is best described abstractly as a pseudorandom function (or more precisely, as a block cipher; we will discuss the difference between these two primitives later in this section).

A passive adversary is one that observes the network traffic and tries to learn something about the underlying messages. It does not alter the ciphertexts. We will define this formally soon.

4.1 *Pseudorandom Functions: Definitions and Basic Properties*

A pseudorandom function family $\mathcal{F} = \{F_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{Y}_n\}_{n\in\mathbb{N}}$ is a family of keyed deterministic functions where, for a random key $k \in \mathcal{K}_n$, the function $F_n(k,\cdot)$ 'looks like' a uniformly random function $f \leftarrow \mathsf{Func}[\mathcal{X}_n, \mathcal{Y}_n]$, where $\mathsf{Func}[\mathcal{X}_n, \mathcal{Y}_n]$ is the set of all functions from $\mathcal{X}_n$ to $\mathcal{Y}_n$. Here, 'looks like' is captured via an indistinguishability game between a challenger and an adversary.

PRF Game

1. Challenger picks $b \leftarrow \{0,1\}$. If $b = 0$, it chooses $k \leftarrow \mathcal{K}_n$, sets $f_0 = F_n(k,\cdot)$. If $b = 1$, it chooses $f_1 \leftarrow \mathsf{Func}[\mathcal{X}_n, \mathcal{Y}_n]$.
2. The adversary makes polynomially many queries. For each query x_i, the challenger sends $f_b(x_i)$.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 9: The security game for defining pseudorandom functions is defined with respect to a function family $\mathcal{F}$ and adversary $\mathcal{A}$. The function F_n takes as input a key $k \in \mathcal{K}_n$, input $x \in \mathcal{X}_n$ and produces an output $\mathcal{Y}_n$.

Note that, in the above game, the challenger is inefficient since representing a uniformly random function requires exponentially many bits. We can also define an equivalent game where the challenger is efficient.

PRF Game: Efficient

1. Challenger picks $b \leftarrow \{0,1\}$ and $k \leftarrow \mathcal{K}_n$. It also maintains a table $\mathcal{T}$ which is originally empty.
2. The adversary makes polynomially many queries. For each query x_i, if $b = 0$, the challenger sends $F_n(k, x_i)$. If $b = 1$, the challenger checks if there is a tuple of the form $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i to the adversary. Else it samples $y_i \leftarrow \mathcal{Y}_n$, adds (x_i, y_i) to $\mathcal{T}$, and sends y_i to the adversary.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 10: The PRF security game, but the challenger defines the uniformly random function 'on-the-fly'.

Definition 4.1. *A keyed function family $\mathcal{F} = \{F_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{Y}_n\}_{n\in\mathbb{N}}$ is a secure pseudorandom function family if, for all p.p.t. adversaries $\mathcal{A}$,*

$$\mathsf{PRFAdv}[\mathcal{A},\mathcal{F}] = \left|\Pr[\mathcal{A} \text{ wins the PRF security game (Figure 10) w.r.t. } \mathcal{F}] - \frac{1}{2}\right|$$

is a negligible function. ◊

Note that there are $|\mathcal{K}_n|$ keys, but $|\mathcal{X}_n|^{|\mathcal{Y}_n|}$ functions mapping $\mathcal{X}_n$ to $\mathcal{Y}_n$. If we take $\mathcal{K}_n = \mathcal{X}_n = \mathcal{Y}_n = \{0,1\}^n$, then the number of keys is 2^n, but the number of functions is 2^{n2^n}, which is much larger than $|\mathcal{K}_n|$.

PRFs imply PRGs

PRFs are widely used in cryptography. In this section, let us see how to use PRFs to build PRGs. Let $F_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. We will use this to build a deterministic function $G_n : \{0,1\}^n \to \{0,1\}^{2n}$, and then prove that $\mathcal{G} = \{G_n\}_{n\in\mathbb{N}}$ is a secure PRG. The following are some constructions that were proposed in class.

Construction 4.2 (PRG from PRFs: Candidate 1)**.**

$$G_n(s) = F_n(0^n, s) \,||\, F_n(1^n, s)$$

In this construction, the PRG output is derived by evaluating the PRF on the same input (which is the PRG input), using two different keys. Note that the keys are not random; they are some fixed strings (0^n and 1^n in this construction).

This construction may not be a secure PRG. At a high level, this is because PRF security is guaranteed only when the PRF key is random. Here, the key is some fixed string. It is possible that F is a secure PRF, but the above construction G is not a secure PRG. For instance, consider a keyed function F' that is a secure PRF. Consider the function F defined as follows:

$$F(k,x) = \begin{cases} 0^n \text{ if } k = 0^n \\ F'(k,x) \text{ otherwise} \end{cases}$$

If F' is a secure PRF, then so is F (prove this using a formal reduction). However, when we use F' to derive a PRG G as above, the PRG will not be secure. For any seed s, the first n bits of $G(s)$ will always be 0^n. ◊

Construction 4.3 (PRG from PRFs: Candidate 2)**.**

$$G_n(s) = F_n(s,s) \,||\, F_n(s, s \oplus 1^n))$$

In this contruction, if s is random, the key is random. However, even this construction is not a secure construction. The issue in this one is a bit more subtle. PRF security says that, for a uniformly random key k, the PRF evaluation on adversarially chosen inputs, looks random. However, the PRF adversary can

never choose the key k as one of its queries. Again, suppose F' is a secure PRF. Consider the following keyed function F:

$$F(k,x) = \begin{cases} 0^n \text{ if } k = x \\ F'(k,x) \text{ otherwise} \end{cases}$$

If F' is a secure PRF, then so is F.[a] *However, when we use F' to derive a PRG G as above, the PRG will not be secure. For any seed s, the first n bits of $G(s)$ will always be 0^n.* ◊

[a] You are encouraged to think about this proof. The proof requires a simple idea which we will discuss later in the course.

Construction 4.4 (PRG from PRFs: Candidate 3)**.**

$$G_n(s) = F_n(s,0^n) \,||\, F_n(s,1^n)$$

In this construction, the PRG seed is used as the key for the PRF, and using this key, it is evaluated on two distinct points (say 0^n and 1^n). This construction is provably secure (prove this using a formal reduction). ◊

Construction 4.5 (PRG from PRFs: Candidate 4)**.**

$$G_n(s) = F_n(s,0^n) \,||\, F_n\left(s, F_n(s,0^n)\right)$$

This construction is also provably secure, we will see its proof in the next section (will be discussed in Lecture 09/10, you are encouraged to think about it and discuss with me). ◊

An alternate security game for PRFs

In the security game for PRFs, the adversary either gets query access to $F(k,\cdot)$, or a truly random function f. What if the adversary also got some 'training queries' where the adversary, on query x, receives $F(k,x)$. Do these training queries help the adversary? We define this security game below.

Definition 4.6. *A keyed function family $\mathcal{F} = \{F_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{Y}_n\}_{n\in\mathbb{N}}$ is a secure pseudorandom function family even with training queries if, for all p.p.t. adversaries $\mathcal{A}$,*

$$\mathsf{PRFAdv}'[\mathcal{A},\mathcal{F}] = \left|\Pr[\mathcal{A} \text{ wins the PRF security game (Figure 11) w.r.t. } \mathcal{F}] - \frac{1}{2}\right|$$

is a negligible function. ◊

Note that the same key k is used for the training queries and the test queries (if challenger chooses $b = 0$). As some of you guessed in class, these training queries are useless (that is, Definition 4.24 and Definition 4.6 are equivalent).

Lecture 07: August 18th, 2023

Claim 4.7. *Let $\mathcal{F}$ be a family of keyed functions with keyspace $\mathcal{K}$, input space $\mathcal{X}$ and output space $\mathcal{Y}$. If there exists a p.p.t. adversary $\mathcal{A}$ that wins the PRF game with training*

PRF Game with training queries

1. Challenger picks $b \leftarrow \{0,1\}$ and $k \leftarrow \mathcal{K}_n$. It also maintains table $\mathcal{T}$, and sets $\mathcal{S}_0$, $\mathcal{S}_1$ which are originally empty. The set $\mathcal{S}_0$ represents the 'test queries' while $\mathcal{S}_1$ represents the 'training queries'.
2. The adversary makes polynomially many queries. Each query is either a 'training query' or a 'test query'.
   - For each 'test query' x_i, the challenger checks if $x_i \in \mathcal{S}_1$ (that is, if x_i was queried as a 'training query'). If so, it outputs $\perp$.
     Else, if $b = 0$, the challenger sends $F_n(k, x_i)$. If $b = 1$, the challenger checks if there is a tuple of the form $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i to the adversary. Else it samples $y_i \leftarrow \mathcal{Y}_n$, adds (x_i, y_i) to $\mathcal{T}$, and sends y_i to the adversary. It also adds x_i to $\mathcal{S}_0$.
   - For each 'training query' x_i, the challenger checks if $x_i \in \mathcal{S}_0$ (that is, if x_i was queried as a 'test query'). If so, it outputs $\perp$.
     Else it sends $F(k, x_i)$ and adds x_1 to $\mathcal{S}_1$.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 11: The PRF security game, but the adversary also gets 'training queries'.

queries (that is, $\mathsf{PRFAdv}'[\mathcal{A}, \mathcal{F}]$ *is non-negligible), then there exists a p.p.t. reduction algorithm* $\mathcal{B}$ *that wins the PRF game without training queries (that is,* $\mathsf{PRFAdv}[\mathcal{B}, \mathcal{F}]$ *is non-negligible).*

Proof. The proof will follow via a hybrid argument. World-0 and World-1 will correspond to the PRF game with training queries, conditioned on $b = 0$ and $b = 1$ respectively.

World 0:

1. Challenger picks $k \leftarrow \mathcal{K}$ and initializes empty sets $\mathcal{S}_0, \mathcal{S}_1$. It responds to the test/training queries as follows.
   - For each test query x, it first checks if $x \in \mathcal{S}_1$. If so, it sends $\perp$.
     Else, it sends $F(k, x)$ and adds x to $\mathcal{S}_0$.
   - For each training query x', it first checks $x' \in S_0$. If so, it sends $\perp$.
     Else it sends $F(k, x')$ and adds x' to $\mathcal{S}_1$.
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_0$$

World 1:

1. Challenger picks $k \leftarrow \mathcal{K}$ and initialises empty table $\mathcal{T}$ and empty sets $\mathcal{S}_0, \mathcal{S}_1$. It responds to the test/training queries as follows.
   - For each test query x, it first checks if $x \in \mathcal{S}_1$. If so, it sends $\perp$.
     Else, it checks if $(x, y) \in \mathcal{T}$. If so, it sends y. If $(x, \cdot) \notin \mathcal{T}$, it samples $y \leftarrow \mathcal{Y}$, sends y and adds (x, y) to $\mathcal{T}$. It also adds x to $\mathcal{S}_0$.
   - For each training query x', it first checks $x' \in \mathcal{S}_0$. If so, it sends $\perp$.
     Else it sends $F(k, x')$ and adds x' to $\mathcal{S}_1$.
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_1$$

Next, we define a hybrid world where the responses to all fresh test/training queries are random.

Hybrid World:

1. Challenger picks $k \leftarrow \mathcal{K}$ and initialises empty tables $\mathcal{T}, \mathcal{T}'$ and empty sets $\mathcal{S}_0, \mathcal{S}_1$. It responds to the test/training queries as follows.
   - For each test query x, it first checks if $x \in \mathcal{S}_1$. If so, it sends $\perp$.
     Else, it checks if $(x, y) \in \mathcal{T}$. If so, it sends y. If $(x, \cdot) \notin \mathcal{T}$, it samples $y \leftarrow \mathcal{Y}$, sends y and adds (x, y) to $\mathcal{T}$. It also adds x to $\mathcal{S}_0$.
   - For each training query x', it first checks $x' \in \mathcal{S}_0$. If so, it sends $\perp$.
     Else it checks if $(x', y') \in \mathcal{T}'$. If so, it sends y'. If $(x', \cdot) \notin \mathcal{T}'$, it samples $y' \leftarrow \mathcal{Y}$, sends y' and adds (x', y') to $\mathcal{T}'$. It also adds x' to $\mathcal{S}_1$.
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb}}$$

Below we present an outline of the two reductions.

Reduction outline: Since $|p_0 - p_1|$ is non-negligible, either $|p_0 - p_{\mathsf{Hyb}}|$ is non-negligible, or $|p_{\mathsf{Hyb}} - p_1|$ is non-negligible. In both these cases, there exists a reduction algorithm $\mathcal{B}$ that wins the PRF game without test queries.

If $|p_0 - p_{\mathsf{Hyb}}|$ is non-negligible, then the reduction algorithm simply forwards all the test/training queries to the PRF challenger, and forwards the responses to $\mathcal{A}$. Note that if the challenger picks $b = 0$, then this corresponds to World-0 for $\mathcal{A}$. If the challenger picks $b = 1$, then this corresponds to the Hybrid World for $\mathcal{A}$.

If $|p_{\mathsf{Hyb}} - p_1|$ is non-negligible, then the reduction forwards all training queries to the PRF challenger, and forwards the responses to $\mathcal{A}$. For the test queries, note that in both worlds (World-1 and Hybrid World), the adversary receives uniformly random strings in response. Therefore, for each fresh test query, the reduction samples a uniformly random string and sends it to $\mathcal{A}$. If the PRF challenger had picked $b = 0$, then this corresponds to World-1, else it corresponds to the Hybrid World.

□

Note: For this proof to work, it is important that the test queries do not overlap with training queries. Where did we use this fact in the above proof?

4.2 *Using PRFs to Build Many-Time Secure Encryption*

Recall, we showed how to use pseudorandom generators to build one-time secure encryption schemes with keys much shorter than the messages. Using pseudorandom functions, we can go beyond one-time security. Before we define many-time security, let us see a few candidate encryption schemes built using secure PRFs.

Construction 4.8 (A stateful encryption scheme)**.** *Let* $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be a secure PRF. Consider the following encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$ *where the encryption algorithm also takes a 'state' as input, and outputs a ciphertext and an updated state.*

- $\mathsf{Enc}(k, m, \mathsf{st})$*: The encryption algorithm outputs* $(\mathsf{st}, m \oplus F(k, \mathsf{st}))$ *as the ciphertext, and* $\mathsf{st}' = \mathsf{st} + 1$ *as the updated state.*

- $\mathsf{Dec}(k, \mathsf{ct} = (\mathsf{st}, c))$*: The decryption algorithm outputs* $c \oplus F(k, \mathsf{st})$.

This is a secure ***stateful*** *encryption scheme. Note that the state is included in the ciphertext, and the updated state is used by the encryption algorithm for encrypting the next message. While such encryption schemes can be used in practice, we would prefer to have stateless encryption.* ◊

Construction 4.9 (A hard-to-decrypt encryption scheme)**.** *Let* $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be a secure PRF. Consider the following encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Enc}(k, m)$*: Output* $m \oplus F(k, m)$.

It is not clear how to decrypt this ciphertext. ◊

(*) Exercise 4.1. *Let* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ *be a secure PRF. Consider the following keyed function* $F' : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$*:*

$$F'(k, x) = x \oplus F(k, x).$$

Show that F' *a secure PRF.*

Construction 4.10 (A deterministic encryption scheme)**.** *Let* $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be a secure PRF. Consider the following encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Enc}(k, m)$*: Output* $m \oplus F(k, k)$.

This is a deterministic encryption scheme, and as we will show later in this section, such schemes cannot satisfy many-time encryption security. ◊

Construction 4.11 (A randomized encryption scheme)**.** *Let* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ *be a secure PRF. Consider the following encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$ *with key space* $\mathcal{K}$*, message space* $\mathcal{Y}$ *and ciphertext space* $\mathcal{X} \times \mathcal{Y}$*:*

- $\mathsf{Enc}(k, m)$*: Sample* $r \leftarrow \mathcal{X}$*, output* $(r, m \oplus F(k, r))$ *as the ciphertext.*
- $\mathsf{Dec}(k, (\mathsf{ct}_1, \mathsf{ct}_2))$*: Output* $\mathsf{ct}_2 \oplus F(k, \mathsf{ct}_1)$.

This is a randomized encryption scheme, and the first encryption scheme where the ciphertext space is larger than the message space. As we will see shortly, both these are necessary for many-time security. ◊

Defining many-time security

We will define many-time security using a security game between an adversary and a challenger. As was discussed in class, there could be multiple ways of

defining many-time security. The following are three candidate security games that were proposed in class.

Many-time security Game: Candidate 1

1. Challenger picks $b \leftarrow \{0,1\}$ and encryption key $k \leftarrow \mathcal{K}$.
2. The adversary sends two vectors of messages (of polynomial length) $\left(\left(m_{i,0}\right)_{i\in[\ell]}, \left(m_{i,1}\right)_{i\in[\ell]}\right)$.
3. For each $i \in [\ell]$, the challenger computes $\mathsf{ct}_i \leftarrow \mathsf{Enc}\left(k, m_{i,b}\right)$ and sends $(\mathsf{ct}_i)_{i\in[\ell]}$.
4. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 12: In this security game, the adversary sends two sets of messages, and the challenger encrypts one of them. The adversary must guess which one was encrypted. This game is stronger than the one-time semantic security game.

Many-time security Game: Candidate 2

1. Challenger picks $b \leftarrow \{0,1\}$ and encryption key $k \leftarrow \mathcal{K}$.
2. The adversary makes polynomially many queries. For the i^{th} query, it sends two messages $\left(m_{i,0}, m_{i,1}\right)$. The challenger computes $\mathsf{ct}_i \leftarrow \mathsf{Enc}\left(k, m_{i,b}\right)$ and sends ct_i.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 13: In this security game, the adversary sends two sets of messages, and the challenger encrypts one of them. However, unlike Game 12, here the adversary can choose the challenge messages adaptively.

Many-time security Game: Candidate 3

1. Challenger picks $b \leftarrow \{0,1\}$ and encryption key $k \leftarrow \mathcal{K}$.
2. The adversary makes polynomially many queries. Each query is either a 'test query' or a 'training query'.
   - For the i^{th} test query, it sends two messages $\left(m_{i,0}, m_{i,1}\right)$. The challenger computes $\mathsf{ct}_i \leftarrow \mathsf{Enc}\left(k, m_{i,b}\right)$ and sends ct_i.
   - For the j^{th} training query, the adversary sends a message m'_j. The challenger computes $\mathsf{ct}'_j \leftarrow \mathsf{Enc}\left(k, m'_j\right)$ and sends ct'_j.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

In the above game, the training queries are allowed to be equal to one of the messages in a test query. This is because we allow the encryption to be randomized.

Figure 14: This is similar to Game 13. Additionally, we also allow the adversary to make training queries.

It is easy to see that Game 14 would result in the strongest definition. But what is the relationship between the other games?

Exercise 4.2. *Show that there exist certain (contrived) encryption schemes where no p.p.t. adversary can succeed in Game 12, but there exists a p.p.t. adversary that can succeed in Game 13.*

(*) Exercise 4.3. *Show that if there exists a p.p.t. adversary $\mathcal{A}$ that can succeed in Game 14 w.r.t. encryption scheme $\mathcal{E}$, then there exists a p.p.t. adversary that can succeed in Game 13 w.r.t. $\mathcal{E}$.*

The adversary is allowed to see encryptions of any messages of its choice (through the 'training queries'), and even gets to choose the challenge plaintexts. Hence the security game is called 'security against chosen plaintext attacks'.

From Exercise 4.3, it follows that Game 13 and 14 can be used for defining many-time security of encryption schemes. This is one of the most widely used security definitions, capturing security against *passive adversaries* — adversaries that monitor the network traffic and aim to learn some information about the underlying messages. [2] This definition is called *many-time security against chosen-plaintext attacks* (or simply CPA security).

Definition 4.12 (Security against chosen plaintext attacks)**.** *An encryption scheme $\mathcal{E}$ is said to be secure against chosen plaintext attacks (or simply CPA secure) if, for any p.p.t. adversary,*

$$\mathsf{CPAAdv}[\mathcal{A},\mathcal{E}] = \left|\Pr[\mathcal{A} \text{ wins Game 13 w.r.t. } \mathcal{E}] - \frac{1}{2}\right|$$

is negligible. ◊

A secure encryption scheme

Lecture 08: August 19th, 2023

In Construction 4.11, we saw a randomized encryption scheme. In this section, we will show that this construction satisfies CPA security, provided F is a secure PRF and the input space of the PRF is sufficiently large.

Before we see the security proof, check that no deterministic encryption scheme can satisfy CPA security. Similarly, no correct encryption scheme whose message space is equal to the ciphertext space can satisfy CPA security. In fact, the size of the ciphertext space must be significantly larger than the size of the message space, as discussed in the exercise below.

Exercise 4.4. *Let $\mathcal{E}$ be an encryption scheme with key space $\{0,1\}^n$, message space $\{0,1\}^{\ell(n)}$ and ciphertext space $\{0,1\}^{\ell(n)+\log n}$. Here, ℓ is some polynomial. Prove that $\mathcal{E}$ cannot satisfy CPA security.* [a]

[a] You may want to use Game 14 for showing an attack. To start, you can work with the case where the message space is $\{0,1\}$, and the ciphertext space is $\{0,1\}^{2+\log n}$.

Claim 4.13. *Let $\mathcal{F} = \{F_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{Y}_n\}_{n\in\mathbb{N}}$ be a family of keyed functions such that $|\mathcal{X}_n|$ is superpolynomial (in n). If there exists a p.p.t. adversary $\mathcal{A}$ that breaks the CPA security of Construction 4.11, then there exists a p.p.t. adversary $\mathcal{B}$ that breaks the PRF security of $\mathcal{F}$.*

For each query $(m_{i,0}, m_{i,1})$, the adversary either receives $(r, m_{i,0} \oplus F(k,r))$, or $(r, m_{i,1} \oplus F(k,r))$. Assuming F is a secure PRF, this is indistinguishable from

[2] It does not capture adversaries that tamper with the network traffic.

the adversary receiving either $(r, m_{i,0} \oplus \mathsf{random}$, or $(r, m_{i,1} \oplus \mathsf{random})$. Note that, by the security of Shannon's OTP, the adversary cannot distinguish between $(r, m_{i,0} \oplus \mathsf{random})$ and $(r, m_{i,1} \oplus \mathsf{random})$. However, it is important that the randomness space is large (else, if two ciphertexts have the same randomness r, then an adversary can win the CPA game). If the randomness space is large, then it is very unlikely that two of the random strings chosen are equal. We formalise this argument below via a hybrid argument.

Proof. Suppose there exists a p.p.t. adversary that breaks the CPA security of Construction 4.11. As in most proofs so far, we start with the two worlds (corresponding to challenger picking $b = 0$ and $b = 1$).

World 0:

1. Challenger picks $k \leftarrow \mathcal{K}$. It responds to the queries as follows.
   - For the i^{th} query $(m_{i,0}, m_{i,1})$, it samples $r_i \leftarrow \mathcal{X}$ and sends $\mathsf{ct}_i = \big(r_i, m_{i,0} \oplus F(k, r_i)\big)$.
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_0$$

World 0:

1. Challenger picks $k \leftarrow \mathcal{K}$. It responds to the queries as follows.
   - For the i^{th} query $(m_{i,0}, m_{i,1})$, it samples $r_i \leftarrow \mathcal{X}$ and sends $\mathsf{ct}_i = \big(r_i, m_{i,1} \oplus F(k, r_i)\big)$.
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_1$$

Next, we define hybrid worlds H_0 and H_1, which are similar to World-0 and World-1, except that the random strings r_i are chosen without replacement.

Hybrid H_0:

1. Challenger picks $k \leftarrow \mathcal{K}$. It responds to the queries as follows.
   - For the i^{th} query $(m_{i,0}, m_{i,1})$, it samples $r_i \leftarrow \mathcal{X}$ uniformly at random without replacement,[a] and sends $\mathsf{ct}_i = \big(r_i, m_{i,0} \oplus F(k, r_i)\big)$.
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},0}$$

[a] In other words, if r_i is equal to one of the previously sampled r_j, it resamples until it gets a fresh sample.

Hybrid H_1:

1. Challenger picks $k \leftarrow \mathcal{K}$. It responds to the test queries as follows.
   - For the i^{th} query $(m_{i,0}, m_{i,1})$, it samples $r_i \leftarrow \mathcal{X}$ uniformly at random without replacement, and sends $\mathsf{ct}_i = \big(r_i, m_{i,1} \oplus F(k, r_i)\big)$.
2. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},1}$$

Finally, we define hybrids H_0' and H_1', where the challenger replaces the PRF evaluations with uniformly random strings.

Hybrid H'_0:

1. Challenger picks $k \leftarrow \mathcal{K}$. It responds to the queries as follows.
   - For the i^{th} query $(m_{i,0}, m_{i,1})$, it samples $r_i \leftarrow \mathcal{X}$ uniformly at random without replacement, $y_i \leftarrow \mathcal{Y}$ and sends $\mathsf{ct}_i = (r_i, m_{i,0} \oplus y_i)$.
2. $\mathcal{A}$ outputs b'.

$$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p'_{\mathsf{Hyb},0}$$

Hybrid H'_1:

1. Challenger picks $k \leftarrow \mathcal{K}$. It responds to the test queries as follows.
   - For the i^{th} query $(m_{i,0}, m_{i,1})$, it samples $r_i \leftarrow \mathcal{X}$ uniformly at random without replacement, $y_i \leftarrow \mathcal{Y}$ and sends $\mathsf{ct}_i = (r_i, m_{i,1} \oplus y_i)$.
2. $\mathcal{A}$ outputs b'.

$$\mathsf{Pr}[\mathcal{A} \text{ outputs } 0] = p'_{\mathsf{Hyb},1}$$

Analysis:

Observation 4.14. *Let t denote the number of queries made by $\mathcal{A}$. Then $\left|p_b - p_{\mathsf{Hyb},b}\right| \leq t^2/|\mathcal{X}_n|$. If $\mathcal{A}$'s running time is polynomial (in n) and $|\mathcal{X}_n|$ is superpolynomial (in n), then $\left|p_b - p_{\mathsf{Hyb},b}\right|$ is negligible.*

Proof.

$$\begin{aligned}
p_b =& \mathsf{Pr}[\mathcal{A} \text{ outputs } 0 \mid \text{Challenger chooses } b] \\
=& \mathsf{Pr}\left[\begin{array}{c} \forall i \neq j, r_i \neq r_j \\ \mathcal{A} \text{ outputs } 0 \end{array} \,\middle|\, \text{Challenger chooses } b\right] \\
&+ \mathsf{Pr}\left[\begin{array}{c} \exists i \neq j, r_i = r_j \\ \mathcal{A} \text{ outputs } 0 \end{array} \,\middle|\, \text{Challenger chooses } b\right] \\
\leq& p_{\mathsf{Hyb},b} + \mathsf{Pr}\left[\exists i \neq j, r_i = r_j \,\middle|\, \text{Challenger chooses } b\right] \\
\leq& p_{\mathsf{Hyb},b} + t^2/|\mathcal{X}_n|
\end{aligned}$$

□

Observation 4.15. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $\left|p_{\mathsf{Hyb},b} - p'_{\mathsf{Hyb},b}\right| = \epsilon$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}$ such that $\mathsf{PRFAdv}[\mathcal{B}, \mathcal{F}] = \epsilon$.*

This follows from the description of hybrids H_b and H'_b. Finally, using the security of Shannon's OTP, we get our final observation.

In case you are not fully comfortable with reductions and security proofs, write a formal reduction-based proof for Observations 4.15, 4.16 and discuss with me after class.

Observation 4.16. $p'_{\mathsf{Hyb},0} = p'_{\mathsf{Hyb},1}$.

Putting everything together, we get

$$p_0 \approx p_{\mathsf{Hyb},0} \approx p'_{\mathsf{Hyb},0} = p'_{\mathsf{Hyb},1} \approx p_{\mathsf{Hyb},1} \approx p_1.$$

□

4.3 *Pseudorandom Permutations (a.k.a. Block Ciphers)*

A closely related cryptographic primitive is the notion of pseudorandom permutations. Similar to PRFs, these are keyed functions, except now all the keyed functions are permutations, and moreover, there exists an efficient inversion

algorithm. More formally, a pseudorandom permutation is a family of keyed functions

$$\mathcal{P} = \left\{ P_n, P_n^{(-1)} : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{X}_n \right\}_{n \in \mathbb{N}}$$

such that both $P_n, P_n^{(-1)}$ are efficiently implementable deterministic functions, and for a random key k, the function $P_n(k, \cdot)$ 'looks like' a random permutation. Let $\mathsf{Perm}[\mathcal{X}]$ denote the set of all permutations with input space $\mathcal{X}$.

Note: P_n is not permuting the bits of the input. Rather, $P(k, \cdot) : \mathcal{X}_n \to \mathcal{X}_n$ is a permutation; that is, for all $x \neq x'$, $P_n(x) \neq P_n(x')$.

PRP Game

1. Challenger picks $b \leftarrow \{0,1\}$. If $b = 0$, it chooses $k \leftarrow \mathcal{K}_n$, sets $f_0 = P_n(k, \cdot)$. If $b = 1$, it chooses $f_1 \leftarrow \mathsf{Perm}[\mathcal{X}_n]$.
2. The adversary makes polynomially many queries. For each query x_i, the challenger sends $f_b(x_i)$.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 15: The security game for defining pseudorandom permutations is defined with respect to a permutation family $\mathcal{P}$ and adversary $\mathcal{A}$.

The most widely used cryptographic circuit, the AES circuit, is assumed to be a pseudorandom permutation. The inversion algorithm provides other ways of building secure encryption, starting with a PRP.

Construction 4.17 (A randomized encryption scheme using a PRP)**.** *Let $P, P^{(-1)} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a secure PRP. Consider the following encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$, message space $\mathcal{X}$ and ciphertext space $\mathcal{X} \times \mathcal{X}$:*

- $\mathsf{Enc}(k, m)$*: Sample $r \leftarrow \mathcal{X}$, output $(r, P(k, r \oplus m))$ as the ciphertext.*
- $\mathsf{Dec}(k, (\mathsf{ct}_1, \mathsf{ct}_2))$*: Output $P^{(-1)}(k, \mathsf{ct}_2) \oplus \mathsf{ct}_1$.*

$\Diamond$

4.4 *Handling Unbounded Length Messages*

In the previous section, we saw encryption schemes with fixed message space (for instance, if we used a PRF with output space $\mathcal{Y}$, then the message space of our encryption scheme was $\mathcal{Y}$). In practice, we would want to encrypt messages of arbitrary length. For concreteness, suppose we want to use AES to build secure encryption schemes. Using the ideas in the previous sections, we will be able to encrypt 128 bit messages. How can we go beyond $\{0,1\}^{128}$?

The first natural idea is to partition our message into blocks of 128 bits, and then encrypt each block separately. This idea is described in Construction 4.18 below.

Construction 4.18 (Encrypting unbounded length messages: Construction 1)**.** *Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a secure PRF. Consider the following encryption scheme with message space $\mathcal{Y}^*$.*

- $\mathsf{Enc}(k, m = (m_1, m_2, \ldots, m_\ell))$: *Sample $r_1, \ldots, r_\ell \leftarrow \mathcal{X}$, output $((r_1, m_1 \oplus F(k, r_1)), \ldots, (r_\ell, m_\ell \oplus F(k, r_\ell)))$.*
- $\mathsf{Dec}\left(k, \mathsf{ct} = \left((\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})_{i \in [\ell]}\right)\right)$: *Let $m_i = \mathsf{ct}_{i,2} \oplus F(k, \mathsf{ct}_{i,1})$. Output $(m_1, \ldots, m_\ell)$.*

The above encryption scheme is provably secure, assuming F is a secure PRF (see Exercise 4.5). However, it is very inefficient in practice — if $\mathcal{X} = \mathcal{Y}$, then the size of the ciphertext is double the size of the message. ◊

(*) **Exercise 4.5.** *Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be a randomized encryption scheme with key space $\mathcal{K}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$ and satisfying CPA security. Construct an encryption scheme with key space $\mathcal{K}$, message space $\mathcal{M}^\ell$ and ciphertext space $\mathcal{C}^\ell$. Prove CPA security of your scheme, assuming $\mathcal{E}$ satisfies CPA security.*

Lecture 09: August 22nd, 2023

We would like to build encryption schemes where the encryption of a message m, of length ℓ, has ciphertext of size ℓ + security parameter. The following candidates were proposed in class. All these are widely used 'encryption modes'.

Strictly speaking, the additive overhead can be reduced to $\omega(\log n)$. Given Exercise 4.4, this is optimal.

Construction 4.19 (Counter (CTR) Mode $\mathcal{E}_{\mathsf{CTR}}$)**.**

- $\mathsf{Enc}(k, m = (m_1, \ldots, m_\ell))$: *The encryption algorithm chooses $r \leftarrow \mathcal{X}$, outputs*

$$\mathsf{ct} = (r, m_1 \oplus F(k, r), m_2 \oplus F(k, r+1), \ldots, m_\ell \oplus F(k, r+\ell-1))$$

- $\mathsf{Dec}(k, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$: *The decryption algorithm outputs*

$$m = (\mathsf{ct}_1 \oplus F(k, \mathsf{ct}_0), \mathsf{ct}_2 \oplus F(k, \mathsf{ct}_0 + 1), \ldots, \mathsf{ct}_\ell \oplus F(k, \mathsf{ct}_0 + \ell - 1))$$

The encryption and decryption can be parallelized. ◊

(*) **Exercise 4.6.** *Show that $\mathcal{E}_{\mathsf{CTR}}$ is CPA secure, assuming $\mathcal{F}$ is a secure PRF and $|\mathcal{X}|$ is superpolynomial in the security parameter.*

Construction 4.20 (Output Feedback (OFB) Mode $\mathcal{E}_{\mathsf{OFB}}$)**.**

- $\mathsf{Enc}(k, m = (m_1, \ldots, m_\ell))$: *The encryption algorithm chooses $r_0 \leftarrow \mathcal{X}$, computes the following for $i = 1$ to ℓ:*

- $r_i = F(k, r_{i-1})$
- $\mathsf{ct}_i = m_i \oplus r_i$.

It outputs $\mathsf{ct} = (r_0, \mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_\ell)$.

- $\mathsf{Dec}(k, \mathsf{ct} = (r_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$: *The decryption algorithm computes*
  - $r_i = F(k, r_{i-1})$
  - $m_i = \mathsf{ct}_i \oplus F(k, r_i)$

  and outputs $m = (m_1, m_2, \ldots, m_\ell)$.

Neither the encryption nor decryption can be parallelized for this scheme. ◊

Construction 4.21 (Cipher Feedback (CFB) Mode $\mathcal{E}_{\mathsf{CFB}}$)**.**

- $\mathsf{Enc}(k, m = (m_1, \ldots, m_\ell))$: *The encryption algorithm chooses* $r \leftarrow \mathcal{X}$, *sets* $\mathsf{ct}_0 = r$, *computes the following for* $i = 1$ *to* ℓ*:*
  - $r_i = F(k, \mathsf{ct}_{i-1})$
  - $\mathsf{ct}_i = m_i \oplus r_i$.

  It outputs $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_\ell)$.
- $\mathsf{Dec}(k, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$: *The decryption algorithm sets* $r_0 = \mathsf{ct}_0$, *computes*
  - $m_i = \mathsf{ct}_i \oplus F(k, \mathsf{ct}_{i-1})$
  - $r_i = F(k, r_{i-1})$

  and outputs $m = (m_1, m_2, \ldots, m_\ell)$.

The decryption can be parallelized for this scheme. ◊

Construction 4.22 (Cipher Block Chaining (CBC) Mode $\mathcal{E}_{\mathsf{CBC}}$)**.**

- $\mathsf{Enc}(k, m = (m_1, \ldots, m_\ell))$: *The encryption algorithm chooses* $r \leftarrow \mathcal{X}$, *sets* $\mathsf{ct}_0 = r$, *computes the following for* $i = 1$ *to* ℓ*:*
  - $\mathsf{ct}_i = P(k, m_i \oplus \mathsf{ct}_{i-1})$

  It outputs $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_\ell)$.
- $\mathsf{Dec}(k, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$: *The decryption algorithm sets* $r_0 = \mathsf{ct}_0$, *computes*
  - $m_i = \mathsf{ct}_{i-1} \oplus P^{(-1)}(k, \mathsf{ct}_i)$

  and outputs $m = (m_1, m_2, \ldots, m_\ell)$.

This is one of the four popular modes where the inversion circuit $P^{(-1)}$ is used. Also, unlike the other modes, here the message needs to be padded if its length is not a multiple of the message block size. The decryption can be parallelized for this scheme. ◊

All these popular modes can proven secure, assuming P is a secure PRF/PRP, and assuming the space of randomness $\mathcal{X}$ is superpolynomial in the security parameter. In these notes, we will prove security of Construction 4.22.

Security Proof for $\mathcal{E}_{\mathsf{OFB}}$

For ease of exposition, we will consider $\mathcal{E}_{\mathsf{OFB}}$ with message space $\mathcal{X}^2$. We will use a PRF/PRP $P : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$.

Consider a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}^2$ defined using P as follows:

$$F(k, x) = (P(k, x), P(k, P(k, x))) \tag{4.23}$$

If we show that F is a pseudorandom function (assuming P is a pseudorandom function), then it follows from Claim 4.13 that $\mathcal{E}_{\mathsf{OFB}}$ is a CPA secure encryption scheme.

Unfortunately, F is not a secure PRF (even if P is a secure PRP). The attack on F works as follows: the adversary can query on any string x, and receives (y_1, y_2). Next, it queries on y_1 and receives (u_1, u_2). If $u_1 = y_2$, then the adversary guesses that the challenger used the pseudorandom function, else it guesses that the challenger picked a truly random function. Check that this attacker wins the PRF security game with probability close to 1.

Weaker definition for pseudorandom functions Since F is not a pseudorandom function, we cannot use Claim 4.13 to conclude that $\mathcal{E}_{\mathsf{OFB}}$ is CPA secure. But can we find a weaker definition for pseudorandom functions such that

1. F satisfies the weaker definition
2. the weaker definition suffices for Claim 4.13

The following weaker definitions were proposed in class:

- *Allow the adversary only one query in the PRF game.* This would ensure that the attack discussed above does not work. However, this definition is too weak, and we cannot use this in Claim 4.13. Recall, in the CPA proof, the number of PRF queries is equal to the number of 'test' queries sent by the CPA adversary. As a result, restricting the number of PRF queries to a single query would mean that we only get semantic security (with a single query), not CPA security.
- *Allow the adversary only non-adaptive queries.* In this weakened definition, the adversary submits all its t queries upfront. The challenger then picks a bit b, and accordingly sends either the PRF evaluations or truly random strings. This definition suffices for Claim 4.13, and F satisfies this definition.

- *Provide PRF evaluations on uniformly random inputs.* Here is a further weakening of the PRF definition: the adversary only gets the evaluations on random inputs chosen by the challenger. In particular, the adversary has no control over the PRF inputs. This weakening is a well-studied primitive called *weak pseudorandom functions*. This notion suffices for Claim 4.13, and the function F satisfies this definition (assuming P is a secure PRF/PRP).

Weak PRF Game

Security game w.r.t. $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

1. The adversary sends t — the number of queries.
2. Challenger picks $b \leftarrow \{0,1\}$ and $k \leftarrow \mathcal{K}_n$.
   It samples t strings $x_i \leftarrow \mathcal{X}$, uniformly at random without replacement.
   If $b = 0$, it sets $y_i = F(k, x_i)$. Else it samples $y_i \leftarrow \mathcal{Y}$ for each $i \in [t]$.
   The challenger sends $((x_i, y_i))_{i \in [t]}$.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 16: The security game for weak PRFs.

Weak Pseudorandom Functions

Definition 4.24. *A keyed function family $\mathcal{F} = \{F_n : \mathcal{K}_n \times \mathcal{X}_n \to \mathcal{Y}_n\}_{n \in \mathbb{N}}$ is a secure weak pseudorandom function family if, for all p.p.t. adversaries $\mathcal{A}$,*

$$\mathsf{wPRFAdv}[\mathcal{A}, \mathcal{F}] = \left| \Pr[\mathcal{A} \text{ wins the weak PRF security game (Figure 16) w.r.t. } \mathcal{F}] - \frac{1}{2} \right|$$

is a negligible function. ◊

To prove CPA security of $\mathcal{E}_{\mathsf{OFB}}$, it suffices to show that weak PRFs are sufficient for Claim 4.13, and that the function F defined in Equation 4.23 is a secure weak PRF (assuming P is a secure PRF).

Observation 4.25. *Assuming F is a weak PRF, Construction 4.11 is a CPA secure encryption scheme.*

The proof of this observation is identical to the proof of Claim 4.13.

Next, we will show that F is a secure weak PRF.

Claim 4.26. *Let $P : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a secure PRF. Then*

$$F(k, x) = (P(k, x), P(k, P(k, x)))$$

is a secure weak PRF.

This proof is not included in the course syllabus. However, if someone can check the calculations below (or suggest simplifications in the proof), I'll get him/her a coffee.

Proof.

World 0:

1. Adversary sends t — number of queries.
2. Challenger picks $k \leftarrow \mathcal{K}$ and samples $x_1, \ldots, x_t \leftarrow \mathcal{X}$, uniformly at random, without replacement. It sets $y_{i,1} = P(k, x_i)$, $y_{i,2} = P(k, y_{i,1})$ and sends $\left((x_i, y_{i,1}, y_{i,2})\right)_{i \in [t]}$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_0$$

World 0:

1. Adversary sends t — number of queries.
2. Challenger picks $k \leftarrow \mathcal{K}$ and samples $x_1, \ldots, x_t \leftarrow \mathcal{X}$, uniformly at random, without replacement. It samples $y_{i,1}, y_{i,2} \leftarrow \mathcal{X}$, and sends $\left((x_i, y_{i,1}, y_{i,2})\right)_{i \in [t]}$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_1$$

A natural first step is the intermediate hybrid where we replace P with a uniformly random function.

Hybrid World:

1. Adversary sends t — number of queries.
2. Challenger picks $f \leftarrow \mathsf{Func}[\mathcal{X}, \mathcal{X}]$ and samples $x_1, \ldots, x_t \leftarrow \mathcal{X}$, uniformly at random, without replacement. It sets $y_{i,1} = f(x_i)$, $y_{i,2} = f(y_{i,1})$ and sends $\left((x_i, y_{i,1}, y_{i,2})\right)_{i \in [t]}$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb}}$$

From the PRF security of P, it follows that $p_0 \approx p_{\mathsf{Hyb}}$. Therefore, to complete the proof, it suffices to show that $p_{\mathsf{Hyb}} \approx p_1$.

$$\begin{aligned}
p_{\mathsf{Hyb}} &= \Pr_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[\; 0 \leftarrow \mathcal{A}\left((x_i, f(x_i), f(f(x_i)))_{i \in [t]}\right) \;\right] \\
&= \Pr_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[\begin{array}{c} 0 \leftarrow \mathcal{A}\left((x_i, f(x_i), f(f(x_i)))_{i \in [t]}\right) \\ \wedge \{ x_1, \ldots, x_t, f(x_1), \ldots, f(x_t) \} \text{ are all distinct} \end{array} \right] \\
&\quad + \Pr_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[\begin{array}{c} 0 \leftarrow \mathcal{A}\left((x_i, f(x_i), f(f(x_i)))_{i \in [t]}\right) \\ \wedge \{ x_1, \ldots, x_t, f(x_1), \ldots, f(x_t) \} \text{ are non-distinct} \end{array} \right] \\
&\leq \Pr_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i, x_i' \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[\; 0 \leftarrow \mathcal{A}\left((x_i, x_i', f(x_i'))_{i \in [t]}\right) \;\right] + \Theta\left(t^2/|X|\right)
\end{aligned}$$

Similarly, we can show that

$$\begin{aligned}
p_{\mathsf{Hyb}} &\geq \Pr_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[\begin{array}{c} 0 \leftarrow \mathcal{A}\left((x_i, f(x_i), f(f(x_i)))_{i \in [t]}\right) \\ \wedge \{ x_1, \ldots, x_t, f(x_1), \ldots, f(x_t) \} \text{ are all distinct} \end{array} \right] \\
&= \Pr_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i, x_i' \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[\; 0 \leftarrow \mathcal{A}\left((x_i, x_i', f(x_i'))_{i \in [t]}\right) \;\right] \left(1 - \Theta\left(t^2/|\mathcal{X}|\right)\right)
\end{aligned}$$

From here, it follows that

(4.27)
$$\left| p_{\mathsf{Hyb}} - \mathsf{Pr}_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i, x_i' \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[0 \leftarrow \mathcal{A}\left(\left(x_i, x_i', f(x_i') \right)_{i \in [t]} \right) \right] \right| \leq \Theta\left(t^2 / |\mathcal{X}| \right)$$

Similarly, we can show that

(4.28)
$$\left| p_1 - \mathsf{Pr}_{\substack{f \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}] \\ x_i, x_i' \leftarrow \mathcal{X} \text{(w.o. rep.)}}} \left[0 \leftarrow \mathcal{A}\left(\left(x_i, x_i', f(x_i') \right)_{i \in [t]} \right) \right] \right| \leq \Theta\left(t^2 / |\mathcal{X}| \right)$$

Combining Equations 4.27, 4.28, we get that $\left| p_{\mathsf{Hyb}} - p_1 \right|$ is negligible if $|\mathcal{X}|$ is superpolynomial. This completes our proof. □

Exercise 4.7. *Recall the following construction of a PRG (Construction 4.5). This construction uses a secure PRF $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.*

$$G(s) = \left(F(s, 0^n), F(s, F(s, 0^n)) \right)$$

Show that G is a secure PRG (assuming F is a secure PRF).

Using the above claim, we have a CPA secure encryption scheme with unbounded message space, and optimal size ciphertexts. A similar security proof can be given for the other modes of encryption.

4.5 *Constructing PRFs and PRPs from Minimal Assumptions*

In Section 3.3, we mentioned that PRGs can be constructed from minimal assumptions (that is, the existence of one-way functions). A similar question can be asked for PRFs and PRPs :

Can we show a generic construction of PRFs/PRPs from one-way functions?

In 1984, Oded Goldreich, Shafi Goldwasser and Silvio Micali [GGM84] showed how to construct PRFs from any length-doubling PRGs. This result, combined with the work of [HILL99], shows that PRFs can also be constructed from minimal assumptions. In 1985, Michael Luby and Charles Rackoff [LR85] showed that any secure PRF can be used to build a secure PRP. Therefore, PRPs can also be constructed generically from one-way functions. We describe both these constructions briefly in this section.

Length-doubling PRGs → PRFs

Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a secure pseudorandom generator. Before we discuss the GGM tree-based construction, let us see a natural candidate, and why such candidates may not work.

- $F(k, x)$: Output $G(k \| x)$. This candidate has key space and input space $\{0,1\}^{n/2}$. It concatenates the key and input, and applies G to the concatenated string. This candidate may not be secure for two reasons:
  - part of the input of G is adversarially chosen; due to this we cannot use the PRG security of G
  - the candidate may not be a weak PRF either, since the adversary receives multiple PRG evaluations that have a related seed (note that for all evaluations, the same first half of the seed is common)

Construction 4.29 (PRF construction using length-doubling PRGs)**.** *The input space of our PRF is $\{0,1\}^{\ell}$, the key space is $\{0,1\}^{n}$ and the output space is $\{0,1\}^{n}$. The PRF evaluation on input x using key k works as follows.*

1. *Let $s = k$. For $i = 1$ to ℓ, do the following:*
   a. *Compute $(s_0, s_1) = G(s)$, where s_0 and s_1 are both n-bit strings.*
   b. *Set $s = s_{x[i]}$.*
2. *Output s.*

◊

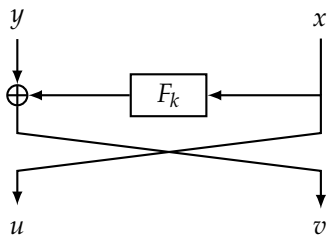
This is a secure PRF, assuming G is a secure length-doubling PRG. The proof goes via a standard hybrid argument for the case where $\ell = O(\log n)$ (recall, we proved a simpler case in Claim 3.17). For $\ell = \omega(\log n)$, it requires a clever hybrid argument; see Theorem 4.10 of the textbook [BS23].

PRFs → PRPs

Lecture 10: August 25th, 2023

Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a secure PRF. Can we use F to build a secure PRP (with appropriate key space, input space and output space)? We start with a few candidates discussed in class.

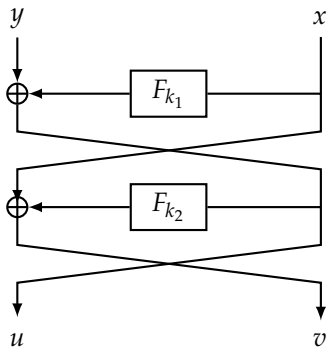
Construction 4.30 (PRP Construction: Attempt 1)**.** *In this attempt, the key space is $\mathcal{K}$, the input and output space is $\mathcal{X} \times \mathcal{Y}$. Check that this is indeed a permutation for every key $k \in \mathcal{K}$.*



It is not a secure PRP as part of the input is included in the output.

◊

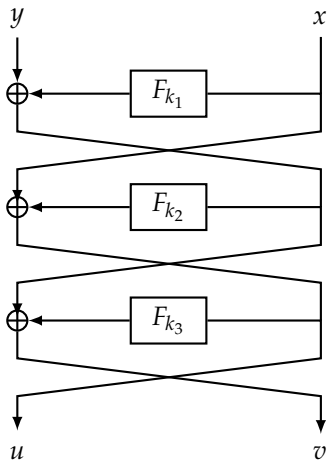
Construction 4.31 (PRP Construction: Attempt 2)**.** *Here we modify Construction 4.30. The key space is $\mathcal{K}^2$, the input and output space is $\mathcal{X} \times \mathcal{Y}$.*



First, check that this is a permutation. This is because the overall function is a composition of permutations. However, it is not a secure PRP. Note that $u = y \oplus F(k_1, x)$. As a result, one can send (x, y_1) and (x, y_2) as queries to the PRP challenger, and if the challenger uses a PRP, the responses (w_1, u_1) and (w_2, u_2) will satisfy $u_1 \oplus u_2 = y_1 \oplus y_2$. If the challenger uses a truly random permutation, then $u_1 \oplus u_2 \neq y_1 \oplus y_2$ with high probability.

However, this construction is a weak pseudorandom permutation *(assuming F is a secure PRF).* ◊

Construction 4.32 (PRP Construction: Attempt 3)**.** *Here we extend Construction 4.31 to three rounds. The key space is $\mathcal{K}^3$, the input and output space is $\mathcal{X} \times \mathcal{Y}$.*



This construction was given by Michael Luby and Charles Rackoff [LR85], and is a provably secure PRP (assuming F is a secure PRF). The first step in the proof is to replace F_{k_1}, F_{k_2} and F_{k_3} with uniformly random functions f_1, f_2, f_3. This follows from the security of F (requires two intermediate hybrids. In the first hybrid world, we replace F_{k_1} with a truly random function f_1, but keep $F_{k_2}, F_{k,3}$. Next, we replace F_{k_2} with a truly random function f_2).

After having replaced F_{k_i} with f_i for all $i \in \{1,2,3\}$, we need to prove that this function is indistinguishable from a uniformly random permutation. This is an information-theoretic argument (that is, does not involve any cryptographic assumptions). A formal proof can be found in [BS23], Theorem 4.9.

One of the students asked whether there is any benefit in going beyond three rounds. The above construction is a secure PRP, but for certain applications, we require that the keyed permutation looks like a random permutation even when the adversary gets access to the inversion function. *This is a stronger security notion called* strong pseudorandom permutations *(see [BS23], Section 4.1.3 and Exercise 4.9).* ◊

4.6 *Constructing PRFs and PRPs in Practice*

The first widely used PRP in practice was the *Data Encryption Standard*.

$$\mathsf{DES}, \mathsf{DES}^{(-1)} : \{0,1\}^{56} \times \{0,1\}^{64} \to \{0,1\}^{64}$$

The DES structure is very similar to the Luby-Rackoff PRP construction. The DES circuit has a *key extension function*, and a keyed function $F : \{0,1\}^{56} \times \{0,1\}^{32} \to \{0,1\}^{32}$. The key extension function is not necessarily a PRG, and F is not necessarily a PRF.

The DES circuit consists of r rounds ($r = 16$ in practice). The key extension function first maps the key $k \in \{0,1\}^{56}$ to r keys $k_1, k_2, \ldots, k_r \in \{0,1\}^{56}$. Next, it parses the input as (x_0, y_0) where $x_0, y_0 \in \{0,1\}^{32}$, and for $i \in [1,r]$, it sets $(x_i, y_i) = (y_{i-1} \oplus F(k_i, x_{i-1}), x_{i-1})$.

See [BS23], Section 4.2.1.1 for more details on how the key extension function, and $F(k,x)$ are implemented. The DES circuit had two shortcomings: short keys and small input space.

The DES was eventually replaced by the *Advanced Encryption Standard* (AES). The AES circuit can be neatly abstracted via a design framework given by Shimon Even and Yishay Mansour. Even and Mansour studied the following question: given a complicated publicly-known permutation π and its inverse $\pi^{(-1)}$, can we construct a PRP using π and $\pi^{(-1)}$? The following candidates were proposed in class.

Construction 4.33 (PRP construction using publicly-known permutation π: Attempt 1)**.** *Given a permutation $\pi : \mathcal{X} \to \mathcal{X}$, let us consider a keyed function $P : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$*

$$P(k,x) = \pi(k \oplus x) \qquad\qquad P^{(-1)}(k,y) = k \oplus \pi^{(-1)}(y)$$

This construction is not a secure PRP. An adversary can query on x, and on receiving y, it computes $k' = \pi^{(-1)}(y) \oplus x$ (recall, π and $\pi^{(-1)}$ are publicly known). The adversary can make a few more queries to check whether the challenger used k' or a truly random permutation. ◊

Construction 4.34 (PRP construction using publicly-known permutation π: Attempt 2)**.** *Given a permutation $\pi : \mathcal{X} \to \mathcal{X}$, let us consider a keyed function $P : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$*

$$P(k,x) = \pi(x) \oplus k \qquad\qquad P^{(-1)}(k,y) = \pi^{(-1)}(k \oplus y)$$

This construction is not a secure PRP. An adversary can query on x, and on receiving y, it computes $k' = \pi(x) \oplus y$. The adversary can make a few more queries to check whether the challenger used k' or a truly random permutation. ◊

Construction 4.35 (PRP construction using publicly-known permutation π: Attempt 3)**.** *Given a permutation $\pi : \mathcal{X} \to \mathcal{X}$, let us consider a keyed function $P : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$*

$$P(k,x) = \pi(x \oplus k) \oplus k \qquad\qquad P^{(-1)}(k,y) = k \oplus \pi^{(-1)}(k \oplus y)$$

Unlike the previous constructions, this one does not have any generic attacks*. An attack is generic if it does not rely on the structure of π, and the adversary requires only input/output access to π and $\pi^{(-1)}$ (check that the attacks on Constructions 4.33,4.34 are generic).* ◊

Capturing generic attacks via idealized security models: Clearly, the security of Construction 4.35 depends on the permutation π used for. However, assuming π is sufficiently 'complicated', we want to know if the construction have any attacks where the adversary can break the PRP by simply studying the input/output behaviour of π? One way to gain some confidence in these constructions is via security proofs in *idealized security models*.

PRP Security in the Ideal Permutation Model

The PRP scheme is defined using a publicly known permutation π, $\pi^{(-1)}$. However, for proving security, we assume the adversary has only oracle access to π and $\pi^{(-1)}$.

1. Challenger picks $b \leftarrow \{0,1\}$ and a uniformly random permutation π. If $b = 0$, it chooses $k \leftarrow \mathcal{K}$, sets $f_0 = P(k,\cdot)$. If $b = 1$, it chooses $f_1 \leftarrow \mathsf{Perm}[\mathcal{X}]$.
2. The adversary makes polynomially many PRP queries, and queries to π and $\pi^{(-1)}$.
   For each PRP query x_i, the challenger sends $f_b(x_i)$.
   For each query x for π evaluation, the challenger sends $\pi(x)$.
   For each query y for $\pi^{(-1)}$ evaluation, the challenger sends $\pi^{(-1)}(y)$.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 17: The security game for defining pseudorandom functions in the ideal permutation model.

The Even-Mansour construction can be proven secure in the ideal permuta-

tion model (see [BS23], Section 4.7.3). We will study other idealized security models later in the course.

Note: A security proof in an idealized security model does not tell us anything about security in the (standard) model. It only rules out generic attacks. However, for many practical constructions, it is the best evidence of security that we have.

(*) Exercise 4.8. *Let p be an n bit prime, $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, and consider permutation $\pi : \mathbb{Z}_p \to \mathbb{Z}_p$:*

$$\pi(0) = 0, \qquad \pi(x) = x^{-1} \bmod p$$

Here $x^{-1} \bmod p$ denotes the unique integer y in $\mathbb{Z}_p$ such that $x \cdot y = 1 \bmod p$. Consider Construction 4.35 with this permutation π, show that it is not a secure PRP. What 'structure/property' of π are you using here?

Back to AES: The AES circuit is an iterated version of Even-Mansour's construction. It uses a permutation $\pi : \{0,1\}^{128} \to \{0,1\}^{128}$ and a key extension function. First, using the key extension function, it maps k to $(k_0, k_2, \ldots, k_9)$. Next, let $x_0 = x$. For $i = 1$ to 9, it computes $x_i = \pi(x_{i-1} \oplus k_{i-1})$ and outputs $y = x_9 \oplus k_9$.

Chapter Summary and References

- We started with a few equivalent security definitions of pseudorandom functions (PRFs). We also saw how to use PRFs to build pseudorandom generators (PRGs), and hence, PRFs can be used to achieve (one-time) semantic security with short keys.
- Next, we discussed a few candidate definitions for many-time security for encryption schemes. This led us to defining security against chosen plaintext attacks (CPA).
- We then showed how to build a CPA secure encryption scheme using PRFs. This scheme had a bounded message space.
- We then talked about pseudorandom permutations (PRPs), a special case of PRFs. Popular encryption algorithms DES and AES are examples of candidate PRPs.
- Next, we saw how to extend the above CPA secure encryption scheme to handle unbounded message spaces. We saw a few popular encryption modes, and discussed a security sketch for one of these modes.
- Finally, we discussed how PRFs and PRPs are built from simpler building blocks, in theory and practice. PRFs can be built from PRGs, and PRPs can be built from PRFs. For the practical candidates, we can prove security in idealized security models.

Textbook References: Boneh-Shoup (v6, [BS23]), Sections 4.1, 4.4, 4.5, 4.6, 4.7, 5.1 - 5.4 **Suggested textbook exercises:** 4.1, 4.2, 4.3, 4.4, 4.6, 4.8, 5.1, 5.4, 5.5, 5.8, 5.13, 5.16

5 Handling Active Attacks via Message Integrity

Lecture 11: August 29th, 2023

In the previous section, we discussed CPA security for encryption schemes, and how to achieve this notion using pseudorandom functions/permutations. However, CPA security only guarantees that a *passive adversary* does not learn anything about the underlying message. In real-world, there are sophisticated attacks that are not captured by CPA security. We discussed malleability attacks in Section 2.3, but there are other, more serious active attacks possible. We start this section with a description of one such attack against the CBC mode of encryption. Next, we will discuss an important cryptographic primitive — message authentication codes (MAC) — that is useful for achieving security against active attacks. We will then discuss the security definition(s) capturing active attacks, followed by constructions of encryption schemes secure against active attacks.

Active attacks on popular modes of encryption

Encryption modes such as CTR, OFB, CFB are immediately susceptible to malleability attacks. In all these modes, the encryption of a message m is $\mathsf{ct} = (\mathsf{ct}_0, m \oplus \mathsf{pseudorandom})$. As a result, an adversary, on receiving ct, can produce a ciphertext which decrypts to $m \oplus r$, for any r of its choice.

In the CBC mode, such a malleability attack may not work. Recall, in the CBC mode, the message m is partitioned into blocks $(m_1, \ldots, m_\ell)$, and the encryption of m is $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ where ct_0 is the randomness used for encryption, $\mathsf{ct}_i = P(k, \mathsf{ct}_{i-1} \oplus m_i)$. Given an encryption of $(m_1, m_2, \ldots, m_\ell)$, an adversary can produce an encryption of $(m_1 \oplus r, m_2, \ldots, m_\ell)$ for any r of its choice. However, given ct, how does the adversary produce an encryption of $(m_1, m_2 \oplus r, m_3, \ldots, m_\ell)$? It is not clear how to execute this malleability attack. However, a clever active attack was shown by Serge Vaudenay [Vau02] in the early 2000s. The attack was applicable on many popular protocols that were using the CBC mode for encryption. We describe one such protocol at a very high level below.

This is an oversimplified description of the protocol.

There are multiple ways to deal with decryption errors, and the protocol specified that the server must output whatever decryption error is produced. Looking ahead, this is important for the attack described below.

There are multiple ways to pad the message, we describe one of them here.

The SSL protocol is an interactive protocol between a client and a server. Both parties share an encryption key. The client encrypts the first message $\mathsf{c\text{-}msg}_1$ and sends it to the server. The server decrypts this ciphertext. If decryption produces an error, then the server outputs this error, else it computes the next message $\mathsf{s\text{-}msg}_1$ (based on $\mathsf{c\text{-}msg}_1$) and sends an encryption of $\mathsf{s\text{-}msg}_1$. Here, the implementation details for encryption and decryption are important.

Encryption details: The messages are byte-strings, while the CBC mode operates in blocks of 16 bytes each. As a result, the last block may need to be padded with zeroes. If the last block consists of t bytes and $t < 16$, then the encryption algorithm appends $16 - t$ bytes. The first appended byte has the number $16 - t$, while the remaining appended bytes are 0. For example, if $\mathsf{c\text{-}msg} = (m[1]\ m[2]\ \ldots\ m[7])$ (where each $m[i]$ is a byte), after padding, the message is $(m[1]\ m[2]\ \ldots m[7]\ 09\ 00\ 00\ \ldots\ 00)$. If the last block consists of 16 bytes, then we add an extra block which has the number 16 in the first byte, followed by fifteen bytes containing 0.

Decryption details: The decryption algorithm first uses the CBC decryption to recover a string s. Next, it checks that the padding is correct (that is, the last

block should end with a number $t \leq 16$ followed by $t-1$ zeroes). If the padding is incorrect, it outputs an error message `'incorrect padding'`. Else, it removes the padding and outputs the resulting string.

We are now ready to describe the attack. At a very high level, the attack intercepts the first ciphertext sent from the client to the server. It then sends multiple tampered ciphertexts to the server, and based on whether it receives a padding error or not, it figures out the padding in the ciphertext. Once it learns the padding, it extracts the message byte-by-byte. For simplicity, we assume that $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$ (that is, the message has only one block).

Learning the padding: The attacker sends $(c_0 \oplus (01\ 00\ \ldots\ 00), c_1)$. If it receives `'incorrect padding'` from the server, then it concludes that the padding is sixteen bytes, else it concludes that the padding is less than sixteen bytes (check why this conclusion is correct). If the attacker did not receive a padding error in the previous iteration, it sends $(c_0 \oplus (00\ 01\ 00\ \ldots\ 00), c_1)$. If it receives a padding error this time, it concludes that the padding is fifteen bytes, else it concludes that the padding is less than fifteen bytes. The attacker continues this way until it receives a padding error, at which point it knows how much padding is present in the ciphertext.

Learning the last byte: Once it learns the padding, the attacker aims to learn the bytes of the message. It starts with the last byte. Roughly, the adversary does the following: given a ciphertext ct which is an encryption of a message with p bytes of padding, it constructs a ciphertext ct' which is an encryption of a message with $p+1$ bytes of padding. That is, suppose it receives an encryption of $(m[1]\ \ldots\ m[7]\ 09\ 00\ \ldots\ 00)$. It transforms this ciphertext to an encryption of $(m[1]\ \ldots\ m[6]\ \alpha\ 00\ 00\ \ldots\ 00)$ for various values of α. Clearly, there is only one value of α for which this results in a non-padding error, and this will be used for learning the last byte.

More formally, given $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, and given that there are p bytes of padding, the adversary does the following for $z = 0$ to $2^{16} - 1$:

- It sets $\mathsf{ct}' = (\mathsf{ct}_0 \oplus (00\ 00\ \ldots\ z\ p\ 00 \ldots 00), \mathsf{ct}_1)$, where z is the byte at position $16 - p$. It sends ct' to the server.
- If it does not receive a padding error, it quits the loop and concludes that the last byte of the message is $z \oplus (p+1)$.

Check that this is indeed the correct value of the last byte. Note that at most 2^{16} calls are required to figure out the last byte, and if the message is ℓ bytes long, then $\ell \cdot 2^{16}$ calls are needed to learn the entire message.

Preventing the padding oracle attack Given this serious attack, the SSL protocol had to be patched. The following fixes were suggested in class:

- Fixing the padding (attempt 1): instead of having the padding at the end of the last message block, have the padding at the start of the first message block. This does not prevent the padding oracle attack. The attack can be suitably modified to work when padding is included at the beginning of the message block.

- Fixing the padding (attempt 2): Suppose p bytes of padding are needed. Instead of padding with p 00 ... 00, pad with r_p 00 ... 00, where r_p is a pseudorandom value derived from p (using a different PRF key k'). This would prevent the padding oracle attack as described above. The adversary will be able to learn how much padding is included, but note that it will not be able to learn the bytes. However, other active attacks might be possible.
- Instead of returning the decryption error in the clear, the server should encrypt the decryption error. Just this will not work, since the size of the ciphertext (containing the string `'padding error'`) could be much smaller than the size of the ciphertext containing s-msg. A natural fix is to pad the error message and then encrypt it. This is indeed what was used in the later versions of SSL/TLS. While this prevents the padding attack, there are more sophisticated *timing attacks* that are still possible (these were shown to be practically feasible too).

The 'correct' way to fix the protocol, and in general, to prevent such active attacks, is to use some cryptographic primitive that ensures that the ciphertext received by the server is indeed the ciphertext that was sent by the client. This property is known as 'ciphertext integrity'. Before we discuss ciphertext integrity, we will discuss how to achieve 'message integrity'. Message integrity ensures that the message received by a receiver is indeed the message sent by the sender. This is the digital analogue of physical signatures. We will discuss the private-key version of this primitive (known as *message authentication codes*) in the following subsections, and the public key analogue (known as *digital signatures*) will be discussed in a later section. After discussing message integrity, we will see how to combine CPA security with message authentication codes/digital signatures in order to obtain ciphertext integrity. It is important to combine them appropriately (the SSL protocol did use message authentication codes, however it was not used properly, and this resulted in the padding oracle attack).

5.1 *Message Authentication Codes: Definition(s)*

Message authentication codes (MACs) are the *private-key digital analogue* of physical signatures. They are a useful building block in cryptography, but can also be used as a stand-alone application. See initial part of Section 6 of [BS23] for a nice introduction to MACs and some motivating applications. It consists of two algorithms: one for signing messages, and another for verifying the signatures. Intuitively, we require that only the authorized party must be able to sign messages. As a result, the signing algorithm must use a secret key. On the other hand, verification may be public. In MACs, the verification is also private (that is, uses the same secret key used for signing the message), but in a later section, we will discuss signature schemes with public verification.

A message authentication code (MAC) for message space $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$, with key space $\{\mathcal{K}_n\}_{n\in\mathbb{N}}$, consists of two polynomial time algorithms (Sign, Verify) with the following syntax and correctness.

Syntax

- $\mathsf{Sign}(k, m)$: takes as input a message $m \in \mathcal{M}_n$, a key $k \in \mathcal{K}_n$, and outputs a signature σ. This algorithm can be deterministic/randomized.

- $\mathsf{Verify}(k, m, \sigma)$: takes as input a message m, signature σ, key k, and outputs either 0 or 1.

Correctness : A MAC $(\mathsf{Sign}, \mathsf{Verify})$ is said to be correct if for all $n \in \mathbb{N}$, keys $k \in \mathcal{K}_n$, messages $m \in \mathcal{M}_n$,

$$\mathsf{Verify}\left(k, m, \mathsf{Sign}(m, k)\right) = 1.$$

The above correctness condition is for schemes with deterministic signing, but a similar correctness condition can be stated for schemes with randomized signing.

Security Definitions for MACs

As usual, whenever we introduce a new cryptographic primitive, we will see various possbile definitions. The first definition (security game described in Figure 18) is a natural security definition: even after seeing many signatures, the adversary should not be able to produce a new signature. This is referred to as 'unforgeability under chosen message attack'. The 'chosen message' comes from the fact that the adversary gets to choose the messages for which it sees a signature.

Security Game for MACs

1. **Setup:** Challenger chooses a signing key k.
2. **Signature queries:** Adversary sends polynomially many signing queries (adaptively). The i^{th} query is a message m_i. The challenger sends $\sigma_i \leftarrow \mathsf{Sign}(k, m_i)$ to the adversary.
   Note that the key k was chosen during setup, and the same key is used for all queries.
3. **Forgery:** The adversary sends a message m^* together with a signature σ^*, and wins if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i, and $\mathsf{Verify}(k, m^*, \sigma^*) = 1$.

Figure 18: Unforgeability under Chosen Message Attack

Definition 5.1 (MACs: Unforgeability under Chosen Message Attacks)**.** *A message authentication code $\mathcal{I} = (\mathsf{Sign}, \mathsf{Verify})$ is said to satisfy* Unforgeability under Chosen Message Attack(UFCMA) *security if, for any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{MACAdv}\left[\mathcal{A}, \mathcal{I}\right] = \mathsf{Pr}\left[\mathcal{A} \textit{ wins the unforgeability game w.r.t. } \mathcal{I}\right]$$

is negligible. ◊

A potentially stronger definition was proposed in class, where the adversary also gets to make verification queries. This security game is defined below.

If every message has a unique signature, then verification queries do not give any additional power to the adversary. ~~For any p.p.t. adversary $\mathcal{A}$ that wins the security game in Figure 19 with probability ϵ using q_s signing queries and q_v verification queries, there exists a p.p.t. reduction $\mathcal{B}$ that wins the security game in Figure 18 with probability ϵ using $q_s + q_v$ signing queries. On receiving a verification query (m, σ), the reduction algorithm $\mathcal{B}$ sends m to the challenger, and receives a signature σ'. If $\sigma = \sigma'$, then it sends 1, else it sends 0.~~

Note that a scheme with deterministic signing may not have unique signatures. (why?)

Security Game for MACs with verification queries

1. **Setup:** Challenger chooses a signing key k.
2. **Queries:** Adversary sends polynomially many queries (adaptively). The queries can either be signing or verification queries, and the challenger uses the same key k (chosen during setup) for responding to these queries.
   - **Signing query:** The i^{th} signing query is a message m_i. The challenger sends $\sigma_i \leftarrow \mathsf{Sign}(k, m_i)$ to the adversary.
   - **Verification query:** The j^{th} verification query is a message m'_j and a signature σ'_j. The challenger sends $\mathsf{Verify}(k, m'_j, \sigma'_j)$ to the adversary.
3. **Forgery:** The adversary sends a message m^* together with a signature σ^*, and wins if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i (each m_i is a signing query), and $\mathsf{Verify}(k, m^*, \sigma^*) = 1$.

Figure 19: Unforgeability under Chosen Message Attack with Verification Queries

However, even if the scheme does not have unique signatures, verification queries do not give the adversary any additional power. For any p.p.t. adversary $\mathcal{A}$ that wins the security game in Figure 19 with probability ϵ using q_s signing queries and q_v verification queries, there exists a p.p.t. reduction $\mathcal{B}$ that wins the security game in Figure 18 with probability ϵ/q_v using at most q_s signing queries. The only way for the adversary to leverage verification queries is by sending an accepting verification query. The reduction can guess the index j which corresponds to the first accepting verification query, and sends '0' in response to all prior verification queries. On receiving the j^{th} verification query, it forwards this as a forgery to the challenger. If the guess is correct, then the reduction wins the security game. A formal argument is presented in Section 6.2 of [BS23].

5.2 *MAC Schemes (for Bounded Message Space)*

Next, we present a few candidate constructions for MACs, using cryptographic primitives that we have seen so far (e.g. PRFs, CPA secure encryption).

Construction 5.2 (MACs from PRFs: via CPA secure encryption)**.** *In this construction, the signing algorithm uses the encryption algorithm, and the verification algorithm uses the decryption algorithm. We discuss this idea using the CPA secure construction from Construction 4.11.*

This construction uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, and has randomized signing. The message space is $\mathcal{Y}$.

- $\mathsf{Sign}(k, m)$: *The signing algorithm chooses $r \leftarrow \mathcal{X}$, outputs $(r, m \oplus F(k, r))$ as the signature.*
- $\mathsf{Verify}(k, m, \sigma = (\sigma_0, \sigma_1))$: *The verification algorithm outputs 1 if $m = \sigma_1 \oplus F(k, \sigma_0)$.*

Given a signature $\sigma = (\sigma_0, \sigma_1)$ on m, an adversary can produce a valid signature on $m \oplus x$ (for any $x \in \mathcal{Y}$) by sending $(\sigma_0, \sigma_1 \oplus x)$. ◊

Construction 5.3 (MACs from PRFs)**.** *This construction also uses PRFs. Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a secure PRF. The message space of our MAC scheme is $\mathcal{X}$.*

- $\mathsf{Sign}(k, m)$: *The signing algorithm outputs $F(k, m)$ as the signature.*
- $\mathsf{Verify}(k, m, \sigma)$: *The verification algorithm outputs 1 if $\sigma = F(k, m)$.*

Note that in this construction, every message has a unique valid signature. This construction is provably secure assuming F is a secure PRF, and $|\mathcal{Y}|$ is superpolynomial in the security parameter. ◊

Lecture 12: September 1st, 2023

Before we get into the security proof of Construction 5.3, note the following two potential vulnerabilities:

- If F is a predictable function (that is, given many values $\{ (x_i, F(k, x_i)) \}_i$, it is easy to compute $F(k, x^*)$ for a new input x^*), then this MAC scheme is not secure.
- If F is a secure PRF, but $|\mathcal{Y}|$ is polynomial in the security parameter, then also the signature scheme is broken. The adversary can simply pick some message m^*, and send a random element of $\mathcal{Y}$ as the signature. The guess will be correct with probability $1/|\mathcal{Y}|$, which is non-negligible if $|\mathcal{Y}|$ is polynomial in n.

The security proof below shows that these are the only possible vulnerabilities.

Claim 5.4. *Assuming F is a secure PRF, and $\mathcal{Y}$ is superpolynomial in the security parameter, Construction 5.3 is an unforgeable MAC scheme.*

Proof Intuition: As a first step, can the adversary win the game without making any queries? In order to do so, the adversary must output $(m^*, F(k, m^*))$. But without making any queries, the adversary has no idea about k, and therefore, no information about $F(k, m^*)$. Its best bet is to pick some m^*, pick a random $y \leftarrow \mathcal{Y}$, and output (m^*, y). The guess will be correct with probability $1/|\mathcal{Y}|$, and therefore, without any queries, the adversary can win with probability at most $1/|\mathcal{Y}|$.

Let us now consider general adversaries. The adversary is receiving some information about the key. But using PRF security, we can replace $F(k, \cdot)$ with a uniformly random function f. As a result, the adversary is playing a modified security game where it interacts with a random function, and must finally output a fresh point m^* together with $f(m^*)$. Since f is a random function, the signing queries provide no information about $f(m^*)$, and the adversary cannot guess $f(m^*)$ with non-negligible probability.

Formal proof: The formal proof will go through a sequence of games, where the original game is the unforgeability game. We will gradually modify the

games, such that the adversary's winning probability only degrades by a negligible additive factor. Finally, we will reach a game where the adversary has little/no chance of winning.

Game 0: This is the original security game.

- **Setup phase:** Challenger chooses a PRF key $k \leftarrow \mathcal{K}$.
- **Query phase:** Adversary can send polynomially many queries. For the i^{th} query m_i, the adversary receives $\sigma_i = F(k, m_i)$.
- **Forgery:** Finally, the adversary must output (m^*, σ^*) such that $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i, and $\sigma^* = F(k, m^*)$.

First, note that for every message, there is exactly one signature that gets accepted (this follows from the construction). As a result, if the adversary sends (m^*, σ^*) that is not equal to any of the (m_i, σ_i) and it also verifies, then $m^* \neq m_i$ for all i.

Let p_0 denote the probability of $\mathcal{A}$ winning Game 0.

Game 1: In this game, the challenger picks a uniformly random function instead of a PRF key.

- **Setup phase:** Challenger chooses a uniformly random function $f \leftarrow \mathsf{Func}[\mathcal{X}, \mathcal{Y}]$.
- **Query phase:** Adversary can send polynomially many queries. For the i^{th} query m_i, the adversary receives $\sigma_i = f(m_i)$.
- **Forgery:** Finally, the adversary must output (m^*, σ^*) such that $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ and $\sigma^* = f(m^*)$.

Let p_1 denote the probability of $\mathcal{A}$ winning Game 1.

Observation 5.5. *Suppose there exists a p.p.t. adversary $\mathcal{A}$ that makes q signature queries, and $|p_0 - p_1| = \epsilon$. Then there exists a p.p.t. algorithm $\mathcal{B}$ that makes $q + 1$ queries to the PRF challenger, and wins the PRF security game with advantage ϵ.*

Proof. The reduction algorithm receives signing queries from $\mathcal{A}$, which it forwards to the PRF challenger. It then forwards the challenger's response to the adversary.

After polynomially many queries, the adversary sends (m^*, σ^*). The reduction algorithm makes a final query to the PRF challenger. It sends m^* to the PRF challenger, gets y^* in response. If $y^* = \sigma^*$, the reduction guesses '0', else it guesses '1'.

If the PRF challenger chose a pseudorandom function, then $\mathcal{B}$ outputs '0' with probability p_0. If the PRF challenger chose a random function, then $\mathcal{B}$ outputs '0' with probability p_1. Hence, its advantage in the PRF game is *epsilon*.

□

Observation 5.6. *For any adversary $\mathcal{A}$, $p_1 = 1/|\mathcal{Y}|$.*

Proof. In Game 1, the challenger chooses a random function f. This is equivalent to choosing the random function 'on-the-fly' as follows: for each new query m_i, the challenger picks a uniformly random string y_i and sends y_i to the adversary. It also maintains a database and adds (m_i, y_i) to the database.

In order to win, the adversary must output (m^*, σ^*) such that $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i and $\sigma^* = f(m^*)$. Note that every message has a unique verifiable signature. Therefore, if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ and σ^* is a valid signature for m^*, then $m^* \neq m_i$ for all i. However, since the random function f is chosen 'on-the-fly', $\sigma^* = f(m^*)$ with probability $1/|\mathcal{Y}|$.

□

Putting together the above claims, we get that if F is a secure PRF, and $1/|\mathcal{Y}|$ is negligible, then the MAC scheme is unforgeable.

The above MAC scheme has message space $\mathcal{X}$. If we use AES, this means the MAC scheme can sign 128 bit messages. In practice, we would like to sign unbounded length messages (suppose you are e-signing a digital document. You wouldn't want the document size to be restricted to 128 bytes). This is discussed in the following subsection.

5.3 *MAC Schemes for Unbounded Message Space*

Recall, when we discussed encryption schemes, there was a simple way to encrypt long messages - split the message into chunks of appropriate size, and encrypt them separately. However, the same approach does not work for MAC schemes. The following are some candidates that were proposed in class. For simplicity, let us first begin with message space that is still bounded, but larger than the PRF input space (we consider $\mathcal{M} = \mathcal{X}^2$ for the initial attempts).

Bounded, but large, message space

Construction 5.7 (MAC Construction: Attempt 1)**.** *The construction uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. The key space of the MAC scheme is $\mathcal{K}$ and the message space is $\mathcal{X}^2$.*

- Sign $(k, m = (m_1, m_2))$*: Output* $\sigma = (F(k, m_1), F(k, m_2))$.
- Verify $(k, m = (m_1, m_2), \sigma = (\sigma_1, \sigma_2))$*: Output* 1 *if* $\sigma_i = F(k, m_i)$ *for* $i \in \{1, 2\}$.

This scheme is not secure; given a signature (σ_1, σ_2) on (m_1, m_2), the adversary can output $m^* = (m_2, m_1)$ and $\sigma^* = (\sigma_2^*, \sigma_1^*)$ as a forgery. ◊

Construction 5.8 (MAC Construction: Attempt 2)**.** *The construction uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. The key space of the MAC scheme is $\mathcal{K}^2$ and the message space is $\mathcal{X}^2$.*

- Sign $(k = (k_1, k_2), m = (m_1, m_2))$*: Output* $\sigma = (F(k_1, m_1), F(k_2, m_2))$.
- Verify $(k = (k_1, k_2), m = (m_1, m_2), \sigma = (\sigma_1, \sigma_2))$*: Output* 1 *if* $\sigma_i = F(k_i, m_i)$ *for* $i \in \{1, 2\}$.

This scheme is not secure due to mix and match attacks*. Given a signature (σ_1^1, σ_1^2) on (m_1, m_1) and a signature (σ_2^1, σ_2^2) on (m_2, m_2), the adversary can*

output $m^* = (m_1, m_2)$ *and* $\sigma^* = (\sigma_1^1, \sigma_2^2)$ *as a forgery.* ◊

Construction 5.9 (MAC Construction: Attempt 3)**.** *This construction is similar to Construction 5.8. The construction uses a PRF* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$*. The key space of the MAC scheme is* $\mathcal{K}^2$*. However, the signing algorithm uses some randomness to update the key used for signing.*

- Sign $(k = (k_1, k_2), m = (m_1, m_2))$*: Choose* $r_1, r_2 \leftarrow \mathcal{K}$*, output* $(r_1, r_2, F(k_1 \oplus r_1, m_1), F(k_2 \oplus r_2, m_2))$*.*
- Verify $(k = (k_1, k_2), m = (m_1, m_2), \sigma = (r_1, r_2, \sigma_1, \sigma_2))$*: Output* 1 *if* $\sigma_i = F(k_i \oplus r_i, m_i)$ *for* $i \in \{1, 2\}$*.*

Mix and match attacks are applicable here as well. Given a signature $(r_1, r_2, \sigma_1^1, \sigma_2^2)$ *on* (m_1, m_1) *and a signature* $(r_3, r_4, \sigma_2^1, \sigma_2^2)$ *on* (m_2, m_2)*, the adversary can output* $m^* = (m_1, m_2)$ *and* $\sigma^* = (r_1, r_4, \sigma_1^1, \sigma_2^2)$ *as a forgery.*
To prevent mix-and-match attacks, it is important to ensure that the first half of one signature cannot be combined with the second half of another signature to produce a new valid signature. Informally, there should be something that 'ties together' all the components of the signature. See Exercise 5.1 below. ◊

Exercise 5.1. *Consider a modification of Construction 5.9. The construction uses a PRF* $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$*. The key space of the MAC scheme is* $\mathcal{K}^2$*, and the message space is* $\mathcal{X}^2$*.*

- Sign $(k = (k_1, k_2), m = (m_1, m_2))$*: Choose* $r \leftarrow \mathcal{K}$*, output* $(r, F(k_1 \oplus r, m_1), F(k_2 \oplus r, m_2))$*.*
- Verify $(k = (k_1, k_2), m = (m_1, m_2), \sigma = (r, \sigma_1, \sigma_2))$*: Output* 1 *if* $\sigma_i = F(k_i \oplus r, m_i)$ *for* $i \in \{1, 2\}$*.*

~~*Show that if F is a secure PRF and* $|\mathcal{Y}|$ *is superpolynomial in the security parameter, then the above construction is a secure MAC.*~~
Show that there exist secure PRFs F such that the above MAC scheme is insecure.

Construction 5.10 (MAC Construction: Attempt 4)**.** *In this construction, we assume* $\mathcal{X} = \{0,1\}^{128}$*, and* $\mathcal{M} = \{0,1\}^{256}$*. We split the message into four blocks, each* 64 *bits.*
The construction uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$*. The key space of the MAC is* $\mathcal{K}$*.*

- Sign$(k, m = (m_1, m_2, m_3, m_4))$*: For* $i = 1$ *to* 4*, let* $\sigma_i = F(k, i \,||\, m_i)$*. Output* $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$*.*
- Verify$(k, m = (m_1, m_2, m_3, m_4), \sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4))$*: Output* 1 *if* $\sigma_i = F(k, i \,||\, m_i)$ *for* $i \in \{1, 2, 3, 4\}$*.*

This attack is also susceptible to mix-and-match attacks. The adversary can

query for a signature on (m_1, m_1, m_1, m_1) and (m_2, m_2, m_2, m_2). Given these two signatures, it can produce a signature on (m_1, m_1, m_2, m_2).

◊

Construction 5.11 (MAC Construction: Attempt 5)**.** *This construction is similar to Construction 5.10. Again, we assume $\mathcal{X} = \{0,1\}^{128}$, and $\mathcal{M} = \{0,1\}^{256}$. We split the message into four blocks, each 64 bits.*
The construction uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. The key space of the MAC is $\mathcal{K}$. The signing algorithm is randomized.

- $\mathsf{Sign}(k, m = (m_1, m_2, m_3, m_4))$*: Choose $r \leftarrow \mathcal{K}$. For $i = 1$ to 4, let $\sigma_i = F(k \oplus r, i \,||\, m_i)$. Output $(r, \sigma_1, \sigma_2, \sigma_3, \sigma_4)$.*
- $\mathsf{Verify}(k, m = (m_1, m_2, m_3, m_4), \sigma = (r, \sigma_1, \sigma_2, \sigma_3, \sigma_4))$*: Output 1 if $\sigma_i = F(k \oplus r, i \,||\, m_i)$ for $i \in \{1,2,3,4\}$.*

Here, the mix-and-match attack is not applicable since the adversary cannot mix two signatures using different values of randomness. However, this construction cannot be proven secure assuming just PRF security of F. This scheme is susceptible to related-key attacks*. Consider a PRF where, for all keys $k \in \mathcal{K}$ and every $x \in \mathcal{X}$, $F(k, x) = F(k \oplus 1^n, x)$. There exist secure PRFs which satisfy this property.*
However, with such a PRF, the above construction will not be secure. An adversary can query on message $m = (m_1, m_2, m_3, m_4)$, and it receives $\sigma = (r, \sigma_1, \sigma_2, \sigma_3, \sigma_4)$. It then sends $m^* = m, \sigma^* = (r \oplus 1^n, \sigma_1, \sigma_2, \sigma_3, \sigma_4)$ as a forgery. Note that $(m, \sigma) \neq (m^*, \sigma^*)$. Also, using the extra property of F, this is a valid signature. Hence unforgeability does not hold, even though F could be a secure PRF. ◊

Construction 5.12 (MAC Construction: Attempt 6)**.** *For this construction, we assume the PRF's input space is $\mathcal{X}' = \mathcal{R} \times \{1, 2, \ldots, \ell\} \times \mathcal{X}$. The construction uses a PRF $F : \mathcal{K} \times \mathcal{X}' \to \mathcal{Y}$. The key space of the MAC is $\mathcal{K}$, and the message space of the MAC is $\mathcal{X}^\ell$. The signing algorithm is randomized.*

- $\mathsf{Sign}(k, m = (m_1, m_2, \ldots, m_\ell))$*: Choose $r \leftarrow \mathcal{K}$. For $i = 1$ to ℓ, let $\sigma_i = F(k, r \,||\, i \,||\, m_i)$. Output $(r, \sigma_1, \sigma_2, \ldots, \sigma_\ell)$.*
- $\mathsf{Verify}\,(k, m = (m_1, m_2, \ldots, m_\ell), \sigma = (r, \sigma_1, \sigma_2, \ldots, \sigma_\ell))$*: Output 1 if $\sigma_i = F(k, r \,||\, i \,||\, m_i)$ for $i \in \{1, 2, \ldots, \ell\}$.*

This construction is provably secure (assuming F is a secure PRF, and $|\mathcal{R}|, |\mathcal{Y}|$ are superpolynomial in the security parameter). In practice, one can use this mode for signing long messages by using $\mathsf{AES} : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$, and taking $\mathcal{R} = \{0,1\}^{64}$, $\mathcal{X} = \{0,1\}^{32}$. It would be secure as long as the number of signatures received by the adversary is significantly less than $\sqrt{|\mathcal{R}|} = 2^{32}$. ◊

Construction 5.13 (MAC Construction: Attempt 7)**.** *This construction uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. The key space of the MAC is $\mathcal{K}$, and the message space of the MAC is $\mathcal{X}^\ell$. The signing algorithm is deterministic.*

- Sign $(k, m = (m_1, m_2, \ldots, m_\ell))$*: Let* $\sigma_0 = \mathbf{0}$*. For* $i = 1$ *to* ℓ*, let* $\sigma_i = F(k, \sigma_{i-1} \oplus m_i)$*. Output* σ_ℓ*.*
- Verify $(k, m = (m_1, m_2, \ldots, m_\ell), \sigma)$*: Output* 1 *if* $\sigma = \sigma_\ell$*, where* $\sigma_0 = \mathbf{0}$ *and* $\sigma_i = F(k, \sigma_{i-1} \oplus m_i)$ *for* $i \in \{1, 2, \ldots, \ell\}$*.*

This construction is provably secure (assuming F is a secure PRF, and $|\mathcal{X}|$ is superpolynomial in the security parameter). An additional advantage (over Construction 5.12) is that it has deterministic signing. However, one downside is that this construction is inherently sequential. ◊

Lecture 13: September 5th, 2023

We will prove security of Construction 5.12. For simplicity, we will assume the message space is $\mathcal{X}^2$, and recall we are given a PRF $F : \mathcal{K} \times \mathcal{X}' \to \mathcal{Y}$ where $\mathcal{X}' = \mathcal{R} \times \{1, 2\} \times \mathcal{X}$. As before, we require $|\mathcal{Y}|$ should be superpolynomial. Additionally, the set $|\mathcal{R}|$ should also be superpolynomial.

Claim 5.14. *Assuming $F : \mathcal{K} \times \mathcal{X}' \to \mathcal{Y}$ is a secure PRF, $|\mathcal{R}|$ and $|\mathcal{Y}|$ are superpolynomial in the security parameter, Construction 5.12 is a secure MAC scheme (as per Definition 5.1).*

Proof. Suppose there exists a p.p.t. adversary $\mathcal{A}$ that wins the MAC game against Construction 5.12 with (non-negligible) probability ϵ. Consider a hybrid-world where the challenger uses a truly random function f instead of a pseudorandom function. This hybrid game is described in detail below.

1. Challenger picks a uniformly random function $f \leftarrow \mathsf{Func}[\mathcal{X}', \mathcal{Y}]$. For the i^{th} signing query $m_i = (m_{i,1}, m_{i,2})$, the challenger does the following:
   (a) it picks $r_i \leftarrow \mathcal{R}$, sets $\sigma_{i,1} = f(r_i \,\|\, 1 \,\|\, m_{i,1})$, $\sigma_{i,2} = f(r_i \,\|\, 2 \,\|\, m_{i,2})$ and sends $(r, \sigma_{i,1}, \sigma_{i,2})$.
2. Adversary sends a message $m^* = (m_1^*, m_2^*)$ and signature $\sigma^* = (r^*, \sigma_1^*, \sigma_2^*)$ and wins if $m^* \neq m_i$ for all i, and $f(r^* \,\|\, 1 \,\|\, m_1^*) = \sigma_1^*$ and $f(r^* \,\|\, 2 \,\|\, m_2^*) = \sigma_2^*$.

Note that, for every message, there is a unique signature that verifies. Hence, if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ and $\mathsf{Verify}(k, m^*, \sigma^*) = 1$, then $m^* \neq m_i$.

Let ϵ' denote the probability of winning in the above hybrid-game. Using the PRF security, we can conclude that $\epsilon \approx \epsilon'$.

Observation 5.15. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $|\epsilon - \epsilon'|$ is non-negligible, then there exists a p.p.t. reduction algorithm $\mathcal{B}$ such that* $\mathsf{PRFAdv}[\mathcal{B}, F] = |\epsilon - \epsilon'|$.

Proof. The reduction algorithm receives signing queries from the adversary $\mathcal{A}$. For each query $m_i = (m_{i,1}, m_{i,2})$, it chooses a uniformly random r_i and sends two queries to the PRF challenger — it sends $(r_i \,\|\, 1 \,\|\, m_{i,1})$ and $(r_i \,\|\, 2 \,\|\, m_{i,2})$ as the PRF queries. It receives $(y_{i,1} \,\|\, y_{i,2})$ from the PRF challenger, and it sends $(r_i, y_{i,1}, y_{i,2})$ to the adversary. Finally, the adversary sends message $m^* = (m_1^*, m_2^*)$ and forgery $(r^*, \sigma_1^*, \sigma_2^*)$. The reduction algorithm sends $(r^* \,\|\, 1 \,\|\, m_1^*)$ and $(r^* \,\|\, 2 \,\|\, m_2^*)$ to the challenger. It gets y_1^*, y_2^* in response, and outputs 0 if $y_1^* = \sigma_1^*$ and $y_2^* = \sigma_2^*$, and $m^* \neq m_i$ for all i.

Check that if the challenger picks a pseudorandom function, then the probability that $\mathcal{B}$ outputs 0 is equal to the probability that $\mathcal{A}$ wins the forgery game in the original game, which is ϵ. On the other hand, if the challenger picks a uniformly random function, then this corresponds to the hybrid-game, and therefore, the probability that $\mathcal{B}$ outputs 0 in this scenario is equal to ϵ'. □

Next, we will show that ϵ' is negligible, assuming $|\mathcal{R}|$ and $|\mathcal{Y}|$ are superpolynomial. As in the proof of Claim 5.4, we will show that in order to win the hybrid game, the adversary must compute f on a fresh input (this was captured via Observation 5.6 in the proof of Claim 5.6). Here we will show that it must compute f on a fresh input with high probability.

Observation 5.16. *For any adversary making at most q signing queries, the probability of winning in hybrid-world is at most $q^2/|\mathcal{R}| + 1/|\mathcal{Y}|$.*

Proof. We are interested in the probability that $\mathcal{A}$ outputs a valid forgery (that is, $m^* \neq m_i$ for all i and $f(r^* \| 1 \| m_1^*) = \sigma_1^*$ and $f(r^* \| 2 \| m_2^*) = \sigma_2^*$). We will break this down into the following cases:

- $\exists i, j$ such that $r_i = r_j$: that is, the randomness used for the i^{th} and j^{th} query are the same. This happens with probability at most $q^2/|\mathcal{R}|$, where q is the total number of queries by $\mathcal{A}$.
- all r_i are distinct, and $r^* \neq r_j$ for all j: in this case, the adversary $\mathcal{A}$ has not received $f(r^* \| 1 \| m_1^*)$ and $f(r^* \| 2 \| m_2^*)$. Therefore, the probability that it correctly guesses the values of $f(r^* \| 1 \| m_1^*)$ and $f(r^* \| 2 \| m_2^*)$ is $1/|\mathcal{Y}|^2$.
- all r_i are distinct, and $r^* = r_j$ for some j: in this case, $m^* \neq m_j$ implies either $m_1^* \neq m_{j,1}$ or $m_2^* \neq m_{j,2}$. Therefore, the adversary has not received either $f(r_j \| 1 \| m_1^*)$ or $f(r_j \| 2 \| m_2^*)$. Hence, the probability that it correctly computes both $f(r_j \| 1 \| m_1^*)$ and $f(r_j \| 2 \| m_2^*)$ is at most $1/|\mathcal{Y}|$.

Putting these together, we can argue that $\epsilon' \leq q^2/|\mathcal{R}| + 1/|\mathcal{Y}|$. □

□

Subtleties in the proof: One needs to be careful with MAC security proofs, especially after we switch to using a uniformly random function (that is, in Observation 5.16). We need to show that a successful forgery implies that f is correctly computed on a new input. Below, we discuss a few of the subtleties:

- suppose we did not assume that all r_i are distinct. In particular, suppose the i^{th} and j^{th} query had the same value r, and suppose the i^{th} signing query was $(m_{i,1}, m_{i,2})$, and the j^{th} signing query was $(m_{j,1}, m_{j,2})$. If $m_{i,1} \neq m_{j,1}$ and $m_{i,2} \neq m_{j,2}$, then the adversary can send $m^* = (m_{i,1}, m_{j,2})$ and $\sigma^* = (r, \sigma_{i,1}, \sigma_{j,2})$ as forgery. Note that, in this case, an adversary wins the forgery game without computing f on a new input.

- suppose our signature was $(r, F(k, r \| m_1), F(k, r \| m_2))$. Again, we cannot conclude that an adversary winning the forgery game implies that f is computed on a new input. For instance, the adversary can query on (m_1, m_2), receives forgery (r, σ_1, σ_2), and sends $m^* = (m_2, m_1)$ and $\sigma^* = (r, \sigma_2, \sigma_1)$ as a forgery. Again, it wins the forgery game without computing f on a new input.

(*) **Exercise 5.2.** *Let $F : \mathcal{K} \times (\mathcal{R} \times \{1,2\} \times \mathcal{X}) \to \mathcal{Y}$ be a secure PRF. Consider the following MAC scheme:*

- $\mathsf{Sign}(k, m = (m_1, m_2))$*: Choose $r \leftarrow \mathcal{K}$. For $i \in \{1,2\}$, let $\sigma_i = F(k, r \| i \| m_i)$. Output $(r, \sigma_1 \oplus \sigma_2)$.*
- $\mathsf{Verify}(k, m = (m_1, m_2), \sigma = (r, \sigma_1))$*: Output 1 if $\sigma_1 = F(k, r \| 1 \| m_1) \oplus F(k, r \| 2 \| m_2)$.*

Show that the above scheme is a secure MAC scheme, assuming $|\mathcal{Y}|$ and $|\mathcal{R}|$ are superpolynomial in the security parameter.

Unbounded message space We saw two constructions (Construction 5.12 and 5.13) that can handle large message spaces. However, for both these constructions, the message space is fixed to be $\mathcal{X}^\ell$ for some ℓ. What if we wanted the message space to be $\mathcal{X}^{\geq 1}$? Unfortunately, both these constructions are not secure if the message space is allowed to be $\mathcal{X}^{\geq 1}$.

Attack on Construction 5.12: The adversary can query for $(m_1, m_2) \in \mathcal{X}$, receives $\sigma^* = (r, \sigma_1, \sigma_2)$ and then sends a forgery (r, σ_1) for $m^* = m_1$.

Attack on Construction 5.13: The adversary can query on $m_1 \in \mathcal{X}$, receives signature σ_1. Next, it queries on $(m_1, m_2) \in \mathcal{X}^2$, receives σ_2. Finally, it sends $m^* = (\sigma_1, m_2)$, and σ_2 as the forgery.

One of the students in class suggested the following fix for Construction 5.13 using randomization.

Construction 5.17 (MAC Construction for Unbounded Message Space: Attempt 1)**.** *This construction uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. The key space of the MAC is $\mathcal{K}$, and the message space of the MAC is $\mathcal{X}^{\geq 1}$. The signing algorithm is randomized.*

- $\mathsf{Sign}\,(k, m = (m_1, m_2, \ldots, m_\ell))$*: Let $\sigma_0 \leftarrow \mathcal{X}$ (that is, σ_0 is sampled uniformly at random from $\mathcal{X}$). For $i = 1$ to ℓ, let $\sigma_i = F(k, \sigma_{i-1} \oplus m_i)$. Output (σ_0, σ_ℓ).*
- $\mathsf{Verify}\,(k, m = (m_1, m_2, \ldots, m_\ell), \sigma = (\sigma_0, \sigma'))$*: Output 1 if $\sigma' = \sigma_\ell$, where $\sigma_i = F(k, \sigma_{i-1} \oplus m_i)$ for $i \in \{1, 2, \ldots, \ell\}$.*

This construction is not secure. An adversary can query for signature on $m \in \mathcal{X}$, receives (σ_0, σ), and sends message $m^* = \sigma_0$ together with (m, σ) as the forgery. ◊

It is possible to fix these schemes to make them secure even when the message space is unbounded. We present those candidates below.

Construction 5.18 (MAC Construction for Unbounded Message Space: Attempt 2)**.** *This construction is an adaptation of Construction 5.12. The only change is that we also include the message length in each input to F. The message space for this construction is $\mathcal{X}^{\leq 2^n}$ which, for all practical purposes, is unbounded (here n is the security parameter).*
For this construction, we assume the PRF's input space is $\mathcal{X}' = \mathcal{R} \times \{1,2,\ldots,2^n\} \times \{1,2,\ldots,\ell\} \times \mathcal{X}$. The construction uses a PRF $F : \mathcal{K} \times \mathcal{X}' \to \mathcal{Y}$.

- $\mathsf{Sign}(k, m = (m_1, m_2, \ldots, m_\ell))$*: Choose* $r \leftarrow \mathcal{K}$*. For* $i = 1$ *to* ℓ*, let* $\sigma_i = F(k, r \| \ell \| i \| m_i)$*. Output* $(r, \sigma_1, \sigma_2, \ldots, \sigma_\ell)$*.*
- $\mathsf{Verify}\,(k, m = (m_1, m_2, \ldots, m_\ell), \sigma = (r, \sigma_1, \sigma_2, \ldots, \sigma_\ell))$*: Output* 1 *if* $\sigma_i = F(k, r \| \ell \| i \| m_i)$ *for* $i \in \{1,2,\ldots,\ell\}$*.*

This construction is provably secure (assuming F is a secure PRF, and $|\mathcal{R}|, |\mathcal{Y}|$ are superpolynomial in the security parameter). ◊

Construction 5.19 (MAC Construction for Unbounded Message Space: Attempt 3)**.** *This construction is an adaptation of Construction 5.13. It uses a PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. The key space of the MAC is $\mathcal{K}^2$, and the message space of the MAC is $\mathcal{X}^{\geq 1}$. The signing algorithm is deterministic.*

- $\mathsf{Sign}\,(k = (k_1, k_2), m = (m_1, m_2, \ldots, m_\ell))$*: Let* $\sigma_0 = \mathbf{0}$*. For* $i = 1$ *to* ℓ*, let* $\sigma_i = F(k_1, \sigma_{i-1} \oplus m_i)$*. Output* $\sigma = F(k_2, \sigma_\ell)$*.*
- $\mathsf{Verify}\,(k = (k_1, k_2), m = (m_1, m_2, \ldots, m_\ell), \sigma)$*: Output* 1 *if* $\sigma = F(k_2, \sigma_\ell)$*, where* $\sigma_0 = \mathbf{0}$ *and* $\sigma_i = F(k_1, \sigma_{i-1} \oplus m_i)$ *for* $i \in \{1,2,\ldots,\ell\}$*.*

This construction is provably secure (assuming F is a secure PRF, and $|\mathcal{X}|$ is superpolynomial in the security parameter). It is popularly referred to as 'encrypted CBC MAC' (because $F(k_2, \sigma_\ell)$ is seen as an encryption of the CBC output). ◊

Lecture 14: September 19th, 2023

Why two keys for ECBC-MAC: Construction 5.19 (popularly referred to as ECBC-MAC) uses two independently chosen PRF keys. One of the students asked in class whether we can use the same PRF key instead of two independently chosen PRF keys. Note that signing a message $(m_1, \ldots, m_\ell)$ using this single-key variant is equivalent to signing $(m_1, \ldots, m_\ell, \mathbf{0})$ using Construction 5.13 (that is, regular CBC-MAC). Note that this would prevent the attack discussed earlier, but there is a different attack that was suggested in class. This attack works as follows: adversary queries on $(x, \mathbf{0}, \mathbf{0})$, receives σ_1. Next it queries on x, receives σ_2. Finally the adversary sends σ_2 as the message, and σ_1 as the forgery. Check that this is indeed a valid forgery.

(*) Exercise 5.3. *Prove that Construction 5.18 is an unforgeable MAC. The proof will be very similar to the proof of Claim 5.14.*

Proving security of ECBC-MAC: One way to prove security of ECBC-MAC is by showing that the resulting function is a PRF. However, that's requires a careful analysis. There is a simpler security proof. ECBC-MAC is a special case of *hash-and-PRF* paradigm (or more generally, *hash-then-sign*), which we discuss below.

5.4 *Hash-then-Sign/Hash-then-PRF*

Note, in the ECBC-MAC construction, we first compress the message $(m_1, \ldots, m_\ell)$ using $F(k_1, \cdot)$, producing a short digest σ_ℓ. Next, we output $F(k_2, \sigma_\ell)$. The computation of σ_ℓ can be seen as a hash computation of the message. What property do we need from this hash computation? At the very least, it should be difficult for an adversary to find two messages that map to the same digest. There are different ways of formalising this, and each leads to a different cryptographic primitive. We require the following properties from this new primitive:

(a) it should be a keyed function that is deterministic and efficiently computable,
(b) it should be compressing,
(c) it should suffice for proving security of construction obtained from the 'hash-then-PRF/hash-then-Sign' paradigm

We will start with the weakest cryptographic primitive satisfying these properties — universal hash functions.

Universal Hash Functions

Definition 5.20. *A keyed function family $\mathcal{H} = \{ H_\lambda : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda \}_{\lambda \in \mathbb{N}}$ is called a universal hash function if each H_λ is efficiently computable, $|\mathcal{X}_\lambda| > |\mathcal{Y}_\lambda|$, and for any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{UHFAdv}[\mathcal{A}, \mathcal{H}] = \mathsf{Pr}_{k \leftarrow \mathcal{K}_\lambda}\left[(x_0, x_1) \leftarrow \mathcal{A}(1^\lambda) \textbf{ and } H(k, x_0) = H(k, x_1) \right]$$

is negligible. ◊

As usual, we will skip the dependence on λ when it is clear from the context. We can view this security definition via a security game between a challenger and an adversary : the challenger picks a key k (but does not send it to the adversary). The adversary sends two strings $x_0, x_1 \in \mathcal{X}$, and wins if $H(k, x_0) = H(k, x_1)$. The pair (x_0, x_1) are called a collision w.r.t. key k if $H(k, x_0) = H(k, x_1)$.

Note that this primitive is very weak — the adversary gets no information about the key, and must find a collision. Therefore, we hope that building this primitive (from one of the standard cryptographic primitives) should be easy. However, does this suffice for the hash-then-PRF paradigm to work? Before we address the latter question, let us see a few candidates for building UHFs. First, note that any PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a UHF, provided $|\mathcal{X}| > |\mathcal{Y}|$ (why?). However,

this observation is not very useful if we want to build a UHF using the AES circuit (since the domain and co-domain are the same for AES). Can we build a UHF with domain $\{0,1\}^{\geq 1}$ using AES?

Construction 5.21 (UHF Construction for Unbounded Message Space: Attempt 1)**.**

This construction uses a PRF $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. The key space of hash function is $\{0,1\}^n$, and the input space is $(\{0,1\}^n)^{\geq 1}$.

$$H(k,(x_1,\ldots,x_\ell)) = \bigoplus_{i=1}^{\ell} F(k,x_i).$$

This construction is not secure. An adversary can pick distinct $x_1, x_2 \in \{0,1\}^n$, and send (x_1,x_2) and (x_2,x_1) as a collision. ◊

Construction 5.22 (UHF Construction for Unbounded Message Space: Attempt 2)**.**

This construction uses a PRF $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. The key space of hash function is $\mathcal{K}$, and the input space is $(\{0,1\}^n)^{\geq 1}$.

$$H(k,(x_1,\ldots,x_\ell)) = \bigoplus_{i=1}^{\ell} F(k+i,x_i).$$

In this construction, we are using related keys, and therefore, it may not be a secure construction (that is, we are assuming some extra properties from the underlying PRF). ◊

Exercise 5.4. *Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRF. Construct a secure PRF F' with appropriate key space, input/output space such that the keyed hash function constructed using F' in Construction 5.22 is not a secure UHF.*

Construction 5.23 (UHF Construction for Unbounded Message Space: Attempt 3)**.**

Let $N = 2^{n/2}$. This construction uses a PRF $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The key space of hash function is $\mathcal{K}$, and the input space is $\left(\{0,1\}^{n/2}\right)^{\leq N}$. While this input space is bounded, the bound is exponential in the security parameter, and hence for all practical purposes, it is unbounded.

$$H(k,(x_1,\ldots,x_\ell)) = \bigoplus_{i=1}^{\ell} F\left(k,(x_i \,\|\, i)\right).$$

This construction is a provably secure UHF. Moreover, it is parallelizable and therefore, practically relevant. ◊

Construction 5.24 (UHF Construction for Unbounded Message Space: Attempt 4)**.**

Somewhat surprisingly, UHFs do not require any cryptographic assumptions at all! Let $p = \Theta\left(2^\lambda\right)$ be a prime number. Consider the following UHF $H_\lambda : \mathbb{Z}_p \times \mathbb{Z}_p^{\geq 1} \to \mathbb{Z}_p$.

$$H_\lambda(k,(x_1,\ldots,x_\ell)) = \left(\sum_{i=1}^{\ell} x_i \cdot k^{i-1}\right) + k^\ell \bmod p.$$

This construction is a provably secure UHF. It uses the input $(x_1,\ldots,x_\ell)$ to define a degree ℓ polynomial $p(y) = \left(\sum_{i=1}^{\ell} x_i \cdot y^i\right) + y^\ell \bmod p$ (the input defines the coefficients of the polynomial). The security proof uses a simple observation: any degree d polynomial over $\mathbb{Z}_p$ can have at most d roots. ◊

(*) Exercise 5.5. *Consider a variant of Construction 5.24, again having the same input/output space:*

$$H_\lambda(k,(x_1,\ldots,x_\ell)) = \left(\sum_{i=1}^{\ell} x_i \cdot k^{i-1}\right) \bmod p.$$

*Show that the above is **not** a secure UHF.*

Lecture 15: September 26th, 2023

Security proofs of UHF candidates

Claim 5.25. *Assuming F is a secure PRF and $|\mathcal{Y}|$ is superpolynomial in the security parameter, Construction 5.23 is a secure UHF.*

Proof. Suppose there exists a p.p.t. adversary $\mathcal{A}$ that wins the UHF game with non-negligible probability ϵ. Let World-0 denote the 'real-world' where the challenger picks PRF key k, adversary sends two messages $x_0 = (x_{0,1},\ldots,x_{0,\ell_0})$, $x_1 = (x_{1,1},\ldots,x_{1,\ell_1})$ such that $x_0 \neq x_1$, and $H(k,x_0) = H(k,x_1)$. That is,

$$\mathsf{Pr}_{\mathcal{A},k\leftarrow\mathcal{K}}\left[\begin{array}{c}(x_0,x_1) \leftarrow \mathcal{A}\left(1^\lambda\right), x_0 \neq x_1 \textbf{ and} \\ \oplus_{i=1}^{\ell_0} F(k,(x_{0,i} \,\|\, i)) = \oplus_{i=1}^{\ell_1} F(k,(x_{1,i} \,\|\, i))\end{array}\right] = \epsilon.$$

Note, in the case of UHFs and MACs, we don't have two indistinguishable worlds. Rather, we start with an attacker on the scheme (we call this the 'real-world'). Next, we gradually change this world, until we reach a hybrid where we are sure the adversary cannot succeed with non-negligible probability.

Here, the probability is over the choice of k and the randomness of $\mathcal{A}$.

Next, we switch the pseudorandom function to a truly random function drawn from $\mathcal{F} = \mathsf{Func}[\{0,1\}^n, \{0,1\}^n]$. Let ϵ' denote the winning probability in this hybrid world. More formally,

$$\mathsf{Pr}_{\mathcal{A}, f \leftarrow \mathcal{F}}\left[\begin{array}{c}(x_0, x_1) \leftarrow \mathcal{A}\left(1^\lambda\right), x_0 \neq x_1 \textbf{ and} \\ \oplus_{i=1}^{\ell_0} f\left((x_{0,i} \,\|\, i)\right) = \oplus_{i=1}^{\ell_1} f\left((x_{1,i} \,\|\, i)\right)\end{array}\right] = \epsilon'.$$

Here the probability is over the choice of f and the randomness of $\mathcal{A}$.

Using the PRF security of F, we can conclude that $\epsilon \approx \epsilon'$ (if $|\epsilon - \epsilon'|$ is non-negligible in the security parameter, then there exists a p.p.t. adversary that breaks the PRF security of F — complete this reduction). Next, we need to show that ϵ' is negligible in the security parameter. We will show that $\epsilon' \leq 1/|\mathcal{Y}|$, and therefore (assuming $|\mathcal{Y}|$ is superpolynomial in the security parameter) ϵ' is negligible.

Observation 5.26.

$$\mathsf{Pr}_{\mathcal{A}, f \leftarrow \mathcal{F}}\left[\begin{array}{c}(x_0, x_1) \leftarrow \mathcal{A}\left(1^\lambda\right), x_0 \neq x_1 \textbf{ \textit{and}} \\ \oplus_{i=1}^{\ell_0} f\left((x_{0,i} \,\|\, i)\right) = \oplus_{i=1}^{\ell_1} f\left((x_{1,i} \,\|\, i)\right)\end{array}\right] \leq \frac{1}{|\mathcal{Y}|}.$$

Proof. We break this down into two cases: either $\ell_0 \neq \ell_1$, or $\ell_0 = \ell_1$. In the first case, assume $\ell_0 > \ell_1$ (the case where $\ell_0 < \ell_1$ can be handled analogously). In this case,

$$\begin{aligned}&\mathsf{Pr}_{\mathcal{A}, f \leftarrow \mathcal{F}}\left[\begin{array}{c}(x_0, x_1) \leftarrow \mathcal{A}\left(1^\lambda\right), x_0 \neq x_1, |x_0| > |x_1| \textbf{ and} \\ f\left((x_{0,\ell_0} \,\|\, \ell_0)\right) = \left(\oplus_{i=1}^{\ell_0 - 1} f\left((x_{0,i} \,\|\, i)\right)\right) \oplus \left(\oplus_{i=1}^{\ell_1} f\left((x_{1,i} \,\|\, i)\right)\right)\end{array}\right] \\ &= \frac{1}{|\mathcal{Y}|} \cdot \mathsf{Pr}_{\mathcal{A}}\left[(x_0, x_1) \leftarrow \mathcal{A}(1^\lambda), x_0 \neq x_1, |x_0| > |x_1|\right]\end{aligned}$$

The last equality follows from the fact that f is a uniformly random function, and the input for f in the LHS (that is, $(x_{0,\ell_0} \,\|\, \ell_0)$) does not appear on the RHS (none of the terms in the RHS end with ℓ_0).

In the second case, when $\ell_0 = \ell_1 = \ell$, then there exists some index $i^* \in [\ell]$ such that $x_{0,i^*} \neq x_{1,i^*}$.

$$\begin{aligned}&\mathsf{Pr}_{\mathcal{A}, f \leftarrow \mathcal{F}}\left[\begin{array}{c}(x_0, x_1) \leftarrow \mathcal{A}\left(1^\lambda\right), x_0 \neq x_1, |x_0| = |x_1| \textbf{ and} \\ f\left((x_{0,\ell_0} \,\|\, \ell_0)\right) = \left(\oplus_{i=1}^{\ell_0 - 1} f\left((x_{0,i} \,\|\, i)\right)\right) \oplus \left(\oplus_{i=1}^{\ell_1} f\left((x_{1,i} \,\|\, i)\right)\right)\end{array}\right] \\ &= \sum_{i^* \in [\ell]} \mathsf{Pr}_{\mathcal{A}, f \leftarrow \mathcal{F}}\left[\begin{array}{c}(x_0, x_1) \leftarrow \mathcal{A}\left(1^\lambda\right), x_0 \neq x_1 \textbf{ and} \\ i^* \text{ is the first index such that } x_{0,i^*} \neq x_{1,i^*} \textbf{ and} \\ f\left((x_{0,i^*} \,\|\, i^*)\right) = \left(\oplus_{i \neq i^*} f\left((x_{0,i} \,\|\, i)\right)\right) \oplus \left(\oplus_{i=1}^{\ell} f\left((x_{1,i} \,\|\, i)\right)\right)\end{array}\right] \\ &= \frac{1}{|\mathcal{Y}|} \cdot \mathsf{Pr}_{\mathcal{A}}\left[(x_0, x_1) \leftarrow \mathcal{A}(1^\lambda), x_0 \neq x_1, |x_0| = |x_1|\right]\end{aligned}$$

The last equality follows from the fact that for any x_0, x_1 such that $|x_0| = |x_1|$ and

$x_{0,i^*} \neq x_{1,i^*}$,

$$\mathsf{Pr}_{f \leftarrow \mathcal{F}}\left[\ f\left((x_{0,i^*} \,\|\, i^*)\right) = \left(\oplus_{i \neq i^*} f\left((x_{0,i} \,\|\, i)\right)\right) \oplus \left(\oplus_{i=1}^{\ell} f\left((x_{1,i} \,\|\, i)\right)\right)\ \right]$$

Therefore, in both cases, the probability is negligible (assuming $|\mathcal{Y}|$ is super-polynomial). □

Using this observation, we conclude that ϵ' must be negligible in the security parameter (and therefore, ϵ must be negligible). □

Next, we will show that UHFs do not require crypto — Construction 5.24 can be proven secure without any cryptographic assumptions.

Claim 5.27. *Construction 5.24 is a secure UHF.*

Proof. In this construction, we need a large prime (must be exponential in the security parameter). The proof of this claim uses the following fact.

Fact 1. *Let p be a prime, and $h(y) = \sum_{i=0}^{d} h_i \cdot y^i$ be any degree d non-zero polynomial (that is, each coefficient $h_i \in \mathbb{Z}_p$ and at least one coefficient is non-zero). Then,*

$$\left|\{\, y \in \mathbb{Z}_p \ :\ h(y) \bmod p = 0 \,\}\right| \leq d.$$

The above fact can be proven using induction on the degree.

Using this fact, we can prove that Construction 5.24 is a secure UHF. For any $x \in \mathbb{Z}_p^{\ell}$, let h_x denote the corresponding degree ℓ polynomial (that is, if $x = (x_1, \ldots, x_\ell)$, then $h_x(y) = y^\ell + \sum_{i=1}^{\ell} x_i \cdot y^{i-1}$).

Fix any $x_0 \in \mathbb{Z}_p^{\ell_0}$, $x_1 \in \mathbb{Z}_p^{\ell_1}$, and for simplicity, let us assume $\ell_0 = \ell_1 = \ell$. Consider the polynomial

$$h_{x_0,x_1}(y) = \sum_{i=1}^{\ell} (x_{0,i} - x_{1,i}) \cdot y^{i-1}.$$

This polynomial is a non-zero polynomial (since $x_0 \neq x_1$), has degree at most $\ell - 1$, and therefore has at most $\ell - 1$ roots. Note that $H(k, x_0) = H(k, x_1)$ if and only if k is a root of h_{x_0,x_1}. Since there are at most $\ell - 1$ roots,

$$\mathsf{Pr}_{k \leftarrow \mathbb{Z}_p}\left[H(k, x_0) = H(k, x_1)\right] \leq \frac{\ell}{p}$$

and ℓ/p is negligible in the security parameter, since ℓ is polynomial in the security parameter, and $p = \Theta(2^\lambda)$.

□

Using UHFs via Hash-then-PRF paradigm

In the previous section, we saw two constructions of UHFs. Here, we will see how to combine a UHF (with unbounded input space) with a PRF to obtain a secure MAC scheme via the 'hash-then-PRF' paradigm.

Claim 5.28. *Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a secure PRF, and $H : \mathcal{K}' \times \mathcal{M} \to \mathcal{X}$ a secure UHF. Consider the following MAC scheme $\mathcal{I} = (\mathsf{Sign}, \mathsf{Verify})$ with key space $\mathcal{K} \times \mathcal{K}'$, message space $\mathcal{M}$:*

- $\mathsf{Sign}((k,k'),m)$*: Compute* $x = H(k',m)$*, output* $\sigma = F(k,x)$.
- $\mathsf{Verify}((k,k'),m,\sigma)$*: Compute* $x = H(k',m)$*, output* 1 *if* $\sigma = F(k,x)$.

Then, assuming $|\mathcal{Y}|$ is superpolynomial in the security parameter, $\mathcal{I}$ is a secure MAC.

Note: It is important to use two independently sampled keys for the PRF and UHF respectively. Otherwise, the proof will not work, and more importantly, for many commonly-used schemes, the scheme becomes totally insecure (e.g. ECBC-MAC).

Proof of Claim 5.28. The proof will go through a sequence of hybrid games, where we alter the winning condition slightly in each game, until we reach a point where the adversary has little/no chance of winning. Each modification will rely on one of the building blocks that we are using.

- Game 0: This corresponds to the UFCMA game.
  - (Setup) Challenger chooses $k \leftarrow \mathcal{K}$ and $k' \leftarrow \mathcal{K}'$.
  - (Signing queries) The adversary queries for polynomially many signature queries. Since this is a MAC scheme with deterministic signing, we assume that all signing queries are distinct. For each query m_i, the challenger computes $x_i = H(k', m_i)$ and outputs $\sigma_i = F(k, x_i)$.
  - (Forgery) After polynomially many signature queries, the adversary sends (m^*, σ^*) and wins if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i, and $\sigma^* = F(k, H(k', m^*))$. Since every message has a unique valid signature, the adversary wins only if $m^* \neq m_i$ for all i and $\sigma^* = F(k, H(k', m^*))$.
- Game 1: In this game, the challenger uses a truly random function instead of a PRF.
  - (Setup) Challenger chooses $k' \leftarrow \mathcal{K}'$ and $f \leftarrow \mathsf{Func}[\mathcal{X}, \mathcal{Y}]$.
  - (Signing queries) The adversary queries for polynomially many signature queries. For each query m_i, the challenger computes $x_i = H(k', m_i)$ and outputs $\sigma_i = f(x_i)$.
  - (Forgery) After polynomially many signature queries, the adversary sends (m^*, σ^*) and wins if $m^* \neq m_i$ for all i, and $\sigma^* = f(H(k', m^*))$.
- Game 2: This game is identical to the previous one. Instead of choosing a uniformly random function, the challenger defines the random function 'on-the-fly'.
  - (Setup) Challenger chooses $k' \leftarrow \mathcal{K}'$ and maintains a table $\mathcal{T}$, which is initially empty.
  - (Signing queries) The adversary queries for polynomially many signature queries. For each query m_i, the challenger computes $x_i = H(k', m_i)$.
    It checks if there exists a pair corresponding to x_i in $\mathcal{T}$. If $(x_i, y_i) \in \mathcal{T}$, it sends y_i to the adversary.
    Else it samples a fresh $y_i \leftarrow \mathcal{Y}$, adds (x_i, y_i) to $\mathcal{T}$ and sends y_i to $\mathcal{A}$.

  - (Forgery) After polynomially many signature queries, the adversary sends (m^*, σ^*) and wins if $m^* \neq m_i$ for all i, and
    - either $(x^*, y^*) \in \mathcal{T}$ where $x^* = H(k', m^*)$ and $\sigma^* = y^*$
    - or $\sigma^* = y^*$ for a uniformly random $y^* \leftarrow \mathcal{Y}$

- Game 3: In this game, the challenger does not maintain a table $\mathcal{T}$. Instead, it sends a uniformly random y_i for each query. In the final forgery check, it chooses a uniformly random y^*.
  - (Setup) Challenger does nothing in setup.
  - (Signing queries) The adversary queries for polynomially many signature queries. For each query m_i, the challenger samples a fresh $y_i \leftarrow \mathcal{Y}$, and sends y_i to $\mathcal{A}$.
  - (Forgery) After polynomially many signature queries, the adversary sends (m^*, σ^*) and wins if $m^* \neq m_i$ for all i, and $\sigma^* = y^*$ for a uniformly random $y^* \leftarrow \mathcal{Y}$.

Analysis: Let p_i denote the winning probability of $\mathcal{A}$ in Game i. From the description of the games, it is clear that the adversary's winning probability in Game 3 is $1/|\mathcal{Y}|$, which is negligible in the security parameter. However, we need to show that the probabilities in the other games are also negligible. First, note that $p_0 \approx p_1$, using the PRF security. Next, note that $p_1 = p_2$ (Game 1 and Game 2 are identical; in one case the entire random function is chosen upfront, while in the other case it is chosen 'on-the-fly'). Therefore, to complete the argument, we need to show that p_2 is negligible if p_3 is negligible (assuming H is a secure UHF).

Observation 5.29. *For any p.p.t. adversary $\mathcal{A}$ that makes q signing queries,*

$$|p_2 - p_3| \leq (q+1)^2 \cdot (\textit{maximum winning probability in the UHF game}).$$

Proof. If we use a information-theoretically secure UHF, then this observation holds for all adversaries (not just computationally bounded ones). For notational convenience, we will set $m^* = m_{q+1}$, $x_{q+1} = H(k', m^*)$. Let Coll denote the event that there exist two distinct indices $i, j \in [q+1]$ such that $x_i = x_j$.

$$\begin{aligned} p_2 &= \mathsf{Pr}\left[\mathcal{A} \text{ wins in Game 2} \wedge \neg\mathsf{Coll}\right] + \mathsf{Pr}\left[\mathcal{A} \text{ wins in Game 2} \wedge \mathsf{Coll}\right] \\ p_3 &= \mathsf{Pr}\left[\mathcal{A} \text{ wins in Game 3} \wedge \neg\mathsf{Coll}\right] + \mathsf{Pr}\left[\mathcal{A} \text{ wins in Game 3} \wedge \mathsf{Coll}\right] \end{aligned}$$

Note that if there is no collision, then Game 2 and Game 3 are identical. Hence,

$$\begin{aligned} p_2 - p_3 &= \mathsf{Pr}\left[\mathcal{A} \text{ wins in Game 2} \wedge \mathsf{Coll}\right] - \mathsf{Pr}\left[\mathcal{A} \text{ wins in Game 3} \wedge \mathsf{Coll}\right] \\ &\leq \mathsf{Pr}\left[\mathsf{Coll}\right] \end{aligned}$$

Suppose $p_2 - p_3$ is non-negligible. Then $\mathcal{A}$ produces a collision with non-negligible probability, and we can construct a reduction algorithm $\mathcal{B}$ that breaks UHF security as follows. $\mathcal{B}$ simply interacts with $\mathcal{A}$ as in Game 2/Game 3, collects all the $q+1$ queries $\{m_1, \ldots, m_{q+1}\}$. Next, it picks a uniformly random pair of messages

from this set and sends them to the UHF challenger. The reduction algorithm wins the UHF game with probability $\mathsf{Pr}[\mathsf{Coll}]/\binom{q+1}{2}$, which is non-negligible if $\mathsf{Pr}[\mathsf{Coll}]$ is non-negligible.

□

Putting everything together: $p_0 \approx p_1$ due to PRF security, $p_1 = p_2$ (the two games are identical), $p_2 \approx p_3$ due to UHF security, and $p_3 = 1/|\mathcal{Y}|$ by definition of Game 3. This concludes our proof. □

Post-proof discussion: At first, Claim 5.28 might look a bit surprising. This is because the UHF security game does not allow any queries, but we must support polynomially many signing queries. Indeed, the above composition uses the structure of the PRF-based MAC. In particular, if we try to combine a UHF with an arbitrary MAC scheme, then the combination may not be a secure MAC scheme. This is because the base MAC scheme's signing algorithm may also reveal the string that it signs. As a result, with sufficiently many queries, the adversary can learn the hash key (for instance, the information-theoretically secure UHF has this liability). In the PRF-based MAC, however, the signing algorithm does not reveal the string that it's giving a signature on, and as a result, the hash digest remains hidden.

> **(*) Exercise 5.6.** *Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRF. Construct a secure MAC scheme $\mathcal{I} = (\mathsf{Sign}, \mathsf{Verify})$ and a secure UHF $\mathcal{H}$ such that the combination of $\mathcal{I}$ and $\mathcal{H}$ does not result in a secure MAC.*

In order to securely combine a hash function with any secure bounded-message-space MAC, we require stronger security requirement from the hash function. One such primitive which is widely used in cryptography is called *collision-resistant hashing*.

Collision Resistant Hash Functions (CRHFs)

A hash function is said to be collision-resistant if no adversary can find a collision, even after seeing the hash key. Note that Construction 5.24 is not a collision-resistant hash function.

Definition 5.30. *A keyed function family $\mathcal{H} = \{ H_\lambda : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda \}_{\lambda \in \mathbb{N}}$ is called a collision-resistant hash function if each H_λ is efficiently computable, $|\mathcal{X}_\lambda| > |\mathcal{Y}_\lambda|$, and for any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{CRHFAdv}[\mathcal{A}, \mathcal{H}] = \mathsf{Pr}_{k \leftarrow \mathcal{K}_\lambda}\left[(x_0, x_1) \leftarrow \mathcal{A}(1^\lambda, k) \textbf{ and } H(k, x_0) = H(k, x_1)\right]$$

is negligible. ◊

First, note that any MAC scheme with message space $\mathcal{M}$ can be combined with a CRHF that maps long messages to $\mathcal{M}$, resulting in a secure MAC scheme that can handle long messages. Intuitively, the resulting MAC scheme is secure, because any forgery for $\mathcal{I}$ either results in a collision for H or a forgery for $\mathcal{I}_{\text{bdd}}$. The formal proof is left as an exercise below.

(*) Exercise 5.7. *Let $\mathcal{I}_{\text{bdd}} = (\mathsf{Sign}_{\text{bdd}}, \mathsf{Verify}_{\text{bdd}})$ be a secure MAC scheme with key space $\mathcal{K}_{\text{bdd}}$, message space $\mathcal{M}$, and let $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{M}$ be a secure CRHF. Consider the following MAC scheme $\mathcal{I} = (\mathsf{Sign}, \mathsf{Verify})$ with key space $\mathcal{K} \times \mathcal{K}_{\text{bdd}}$ and message space $\{0,1\}^*$.*

- $\mathsf{Sign}((k, k_{\text{bdd}}), m)$*: Compute* $y = H(k, m)$*, and output* $\mathsf{Sign}_{\text{bdd}}(k_{\text{bdd}}, y)$.
- $\mathsf{Verify}((k, k_{\text{bdd}}), m, \sigma)$*: Output* $\mathsf{Verify}_{\text{bdd}}(k_{\text{bdd}}, H(k, m), \sigma)$.

Show that if H is a secure CRHF, and $\mathcal{I}_{\text{bdd}}$ is a secure MAC scheme, then so is $\mathcal{I}$.

(*) Exercise 5.8. *Consider the following modification, where the hash key is chosen during signing each time. More formally, let $\mathcal{I}_{\text{bdd}} = (\mathsf{Sign}_{\text{bdd}}, \mathsf{Verify}_{\text{bdd}})$ be a secure MAC scheme with key space $\mathcal{K}_{\text{bdd}}$, message space $\mathcal{M}$, and let $H : \mathcal{K} \times \{0,1\}^* \to \mathcal{M}$ be a secure CRHF. Consider the following MAC scheme $\mathcal{I} = (\mathsf{Sign}, \mathsf{Verify})$ with key space $\mathcal{K}_{\text{bdd}}$ and message space $\{0,1\}^*$.*

- $\mathsf{Sign}(k_{\text{bdd}}, m)$*: Choose a uniformly random hash key* $k \leftarrow \mathcal{K}$*, compute* $y = H(k, m)$*, and output* $(k, \mathsf{Sign}_{\text{bdd}}(k_{\text{bdd}}, y))$.
- $\mathsf{Verify}(k_{\text{bdd}}, m, (k, \sigma))$*: Output* $\mathsf{Verify}_{\text{bdd}}(k_{\text{bdd}}, H(k, m), \sigma)$.

This transformation may not be secure, even if H is a secure CRHF, and $\mathcal{I}_{\text{bdd}}$ is a secure MAC scheme. Why?

Currently, we don't have any provably secure constructions based on one-way functions/PRFs. However, since CRHFs are immensely useful in practice, we have efficient constructions that have been extensively studied, and are therefore believed to be secure candidates for CRHFs. We will discuss CRHFs in detail in a later section, when we study (public key) digital signatures.

5.5 *Encryption Schemes Secure against Active Attacks*

Lecture 16: September 29th, 2023

We are now ready to discuss how to define and handle active attacks against encryption schemes. Recall, so far in the course, we discussed two broad classes of active attacks. The first was *malleability attacks* (given an encryption of m, the adversary is able to produce encryption of a new message m'), and next we talked about *padding oracle attacks*. The padding oracle attack is a special case of attacks where the adversary obtains some information via decryption queries. Such information leak is hard to prevent in real-world scenarios. In order to capture the information-leak possible from decryption queries, we allow the adversary full access to the decryption functionality. The adversary can query the decryption oracle on any ciphertext of its choice, except the challenge ciphertext. These attacks are referred to as *chosen-ciphertext attacks*, and formally, this is captured via a security game between a challenger and an adversary (the game is described in Figure 20).

This security game captures padding oracle attacks. Note that even malleability attacks are a special case of chosen ciphertext attacks. If an encryption

Chosen Ciphertext Attacks (CCA)

- Challenger picks encryption key $k \leftarrow \mathcal{K}$ and bit $b \leftarrow \{0,1\}$.
- Adversary is allowed polynomially many queries, each query is either an encryption query or a decryption query. The ciphertext sent as decryption query must not be equal to one of the challenge ciphertexts.
  - *Encryption Query:* Adversary sends a pair of challenge messages $(m_{i,0}, m_{i,1})$ and receives challenge ciphertext $\mathsf{ct}_i \leftarrow \mathsf{Enc}(k, m_{i,b})$.
  - *Decryption Query:* Adversary sends a ciphertext ct_j not equal to any of the challenge ciphertexts, and receives $y_j \leftarrow \mathsf{Dec}(k, \mathsf{ct}_j)$.
- The adversary finally outputs its guess b', and wins the game if $b = b'$.

Figure 20: The security game for capturing chosen ciphertext attacks. This is defined with respect to an encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$. The game is similar to the CPA security game, except that the adversary also gets to make decryption queries.

scheme is secure against chosen ciphertext attacks, then no p.p.t. adversary can tamper the encryption of m to produce an encryption of $m \oplus \alpha$ (here α is some adversarially chosen string, but the adversary does not know m or the encryption key).

Here, we are discussing a simple malleability attack that maps m to $m \oplus \alpha$. However, the argument holds for more general malleability attacks too.

Suppose an adversary could do this efficiently. Then we can use this adversary to launch a chosen ciphertext attack. The reduction algorithm sends two distinct messages m_0, m_1. It receives a challenge ciphertext ct^*. It then uses the adversary to alter ct^*, resulting in an encryption of either $m_0 \oplus \alpha$, or $m_1 \oplus \alpha$. Let ct' be the tampered ciphertext. The reduction sends this tampered ciphertext as a decryption query, and receives $m_b \oplus \alpha$. Given this information, the reduction algorithm now knows b.

(*) Exercise 5.9. *Let $(\mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme secure against chosen ciphertext attacks. Then no p.p.t. adversary, given encryptions of m_1 and m_2, can produce an encryption of $m_1 \oplus m_2$. (Note, in this attack scenario, the adversary does not know m_1, m_2 or the encryption/decryption key).*

Authenticated Encryption (AE): Syntax and Security Definition

All the active attacks arise from the following vulnerability: Alice and Bob are communicating using a shared secret key k. Alice receives a ciphertext, but there is no way for Alice to check whether this ciphertext indeed came from Bob. This problem is very similar to the message integrity problem that we considered earlier, and indeed, the security guarantee that we require is *ciphertext integrity*.

Before we discuss ciphertext integrity, we will make a small change to the syntax of Dec. The decryption algorithm either maps the ciphertext to some message in the message space, or outputs a special symbol $\perp$, indicating that decryption failed. That is, $\mathsf{Dec}(k, \mathsf{ct}) \rightarrow y \in \mathcal{M} \cup \{\perp\}$.

Roughly, ciphertext integrity states that an adversary should not be able to produce a valid ciphertext on its own, even after seeing many valid ciphertexts.

As usual, we will allow the adversary to query for encryptions of messages of its choice, and it must produce a new ciphertext whose decryption is not $\perp$. This is described formally in the following security game (see Figure 21).

CT-INT

- (Setup Phase) Challenger chooses an encryption key k.
- (Encryption Queries) Adversary sends polynomially many encryption queries. For the i^{th} query message m_i, it receives $\mathsf{ct}_i \leftarrow \mathsf{Enc}(k, m_i)$.
- (Integrity Attack) Adversary finally outputs a ciphertext ct^* and wins if $\mathsf{Dec}(k, \mathsf{ct}^*) \neq \perp$, and $\mathsf{ct}^* \notin \{\mathsf{ct}_i\}_i$.

Figure 21: Security game for capturing ciphertext integrity attacks

Definition 5.31. *An encryption scheme $\mathcal{E}$ is said to satisfy ciphertext integrity if, for any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{CtIntAdv}[\mathcal{A}, \mathcal{E}] = \Pr[\mathcal{A} \text{ wins the ciphertext integrity game w.r.t. } \mathcal{E}]$$

is negligible. ◊

An encryption scheme that ensures both message confidentiality (via CPA security) and message integrity (via ciphertext integrity) is called a secure *authenticated encryption* scheme.

Any encryption scheme used in practice must be a secure authenticated encryption scheme.

Definition 5.32. *An encryption scheme $\mathcal{E}$ is a secure authenticated encryption scheme if it satisfies CPA security (Definition 4.12) and ciphertext integrity (Definition 5.31).* ◊

There are two important questions at this point: how do we construct encryption schemes that satisfy Definition 5.32, and why does authenticated encryption guarantee security against active attacks? In particular, does a secure authenticated encryption scheme guarantee that there will be no chosen ciphertext attacks? We will address these questions in the following subsections.

Authenticated Encryption Prevents Chosen Ciphertext Attacks

Let $(\mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme that satisfies both semantic security and ciphertext integrity. In this section, we will show that the scheme is also CCA secure. To argue CCA security, the reduction algorithm must handle decryption queries. If the adversary sends a fresh ciphertext for decryption, this ciphertext must be an invalid ciphertext (otherwise the scheme does not satisfy ciphertext integrity).

Lemma 5.33. *Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme that satisfies both semantic security and ciphertext integrity. Then, for every p.p.t. adversary $\mathcal{A}$,*

$$\Pr[\mathcal{A} \text{ wins the CCA security game against } \mathcal{E}]$$

is negligible.

Proof. As usual, we will start with the CCA security game, and alter it gradually, until we reach a game where the adversary has negligible advantage.

- **Game 0:** This is the CCA game against $\mathcal{E}$.
  - *Setup phase* Challenger chooses a key k and a bit b.
  - *Query phase* Adversary can make polynomially many queries.
    For each encryption query $(m_{i,0}, m_{i,1})$, the challenger sends $\mathsf{ct}_i \leftarrow \mathsf{Enc}(k, m_{i,b})$.
    For each decryption query ct'_i (not equal to any of the encryption query responses), the challenger sends $y_i = \mathsf{Dec}(k, \mathsf{ct}'_i)$.
  - *Guess* Adversary sends its guess b' and wins if $b = b'$.
- **Game 1:** This game is similar to the previous one, except that the challenger does not use the secret key for decryption. Instead, it sends $\perp$.
  - *Setup phase* Challenger chooses a key k and a bit b.
  - *Query phase* Adversary can make polynomially many queries.
    For each encryption query $(m_{i,0}, m_{i,1})$, the challenger sends $\mathsf{ct}_i \leftarrow \mathsf{Enc}(k, m_{i,b})$.
    For each decryption query ct'_i (not equal to any of the encryption query responses), the challenger sends $\perp$.
  - *Guess* Adversary sends its guess b' and wins if $b = b'$.

Let p_b be the winning probability in Game b.

Claim 5.34. *If there exists a p.p.t. adversary $\mathcal{A}$ that makes Q decryption queries, and $p_0 - p_1$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}$ that wins the ciphertext integrity game with probability $(p_0 - p_1)/Q$.*

Proof Sketch: The only difference in the two games is if one of the decryption queries is a fresh ciphertext that decrypts to a non-$\perp$ value. The reduction algorithm can guess the first location i^* of such a decryption query. This guess is correct with probability $1/Q$. For all the encryption queries $(m_{i,0}, m_{i,1})$ before this one, the reduction simply forwards $m_{i,b}$ to the ciphertext integrity challenger. For all decryption queries before this one, it sends $\perp$. Finally, on receiving the $i^{*\text{th}}$ decryption query, it forwards this query to the ciphertext integrity challenger.

Claim 5.35. *There exists a p.p.t. algorithm $\mathcal{B}$ that wins the semantic security game with probability p_1.*

Proof sketch: In Game 1, the decryption queries are handled without the secret key. Therefore, the reduction algorithm can interact with the semantic security challenger and the CCA adversary. For the encryption queries, it forwards them to the challenger, and forwards the responses to $\mathcal{A}$. For the decryption queries, it outputs $\perp$.

From Claim 5.35, it follows that p_1 is at most $1/2 + \text{negl}$. From Claim 5.34, it follows that p_0 is negligibly close to p_1, and hence at most $1/2 + \text{negl}$.

□

CPA + MAC → AE

An authenticated encryption scheme can be constructed by appropriately combining a CPA secure encryption scheme together with a MAC scheme. Before discussing the 'provably secure' combination, let us look at a new insecure choices.

Encrypt and MAC. In this approach, the secret key is a MAC key k_{mac} and an encryption key k_{cpa}. The encryption of a message m using $(k_{\text{mac}}, k_{\text{cpa}})$ is $(\mathsf{Sign}(k_{\text{mac}}, m), \mathsf{Enc}(k_{\text{cpa}}, m))$. The decryption algorithm, on receiving (σ, ct), first decrypts ct to recover m. It then checks if $\mathsf{Verify}(k_{\text{mac}}, m, \sigma) = 1$. If verification passes, then it outputs m, else it outputs $\perp$.

This combination neither guarantees ciphertext integrity, nor CPA security.

Why ciphertext integrity does not hold: As suggested by one of the students in class, if the signing algorithm is randomized, then there exists a simple 'mix-and-match' attack. An adversary can pick an arbitrary message m and query for encryption of m twice. Let $(\sigma_1, \mathsf{ct}_1)$ be the first response, and $(\sigma_2, \mathsf{ct}_2)$ be the second response. The adversary sends $(\sigma_1, \mathsf{ct}_2)$ as the attack. With high probability, $\sigma_1 \neq \sigma_2$, and therefore this is a valid attack.

Suppose the base encryption scheme is malleable. Given an encryption of a message, it is possible to produce another encryption of the same message. Then ciphertext integrity can be broken using just one query. The adversary queries for encryption of a message m and receives (σ, ct). It then alters the ciphertext ct to produce a new ciphertext ct' that is also an encryption of m. It finally sends (σ, ct') as the attack on ciphertext integrity.

Why CPA security does not hold: A secure MAC scheme need not hide the underlying message. As a result, the signature can reveal the underlying message.

MAC then Encrypt. The next natural candidate is 'MAC-then-encrypt'.

Construction 5.36 (MAC-then-Encrypt)**.** *Given a MAC scheme* $(\mathsf{Sign}, \mathsf{Verify})$ *with key space* $\mathcal{K}_{\text{mac}}$ *and a CPA secure encryption scheme* $(\mathsf{Enc}_{\text{cpa}}, \mathsf{Dec}_{\text{cpa}})$ *with key space* $\mathcal{K}_{\text{cpa}}$*, consider the following candidate for an authenticated encryption scheme, with key space* $\mathcal{K}_{\text{mac}} \times \mathcal{K}_{\text{cpa}}$*:*

- $\mathsf{Enc}((k_{\text{mac}}, k_{\text{cpa}}), m)$: *Compute* $\sigma \leftarrow \mathsf{Sign}(k_{\text{mac}}, m)$*, and* $\mathsf{ct} \leftarrow \mathsf{Enc}_{\text{cpa}}(k_{\text{cpa}}, m \,\|\, \sigma)$.
- $\mathsf{Dec}((k_{\text{mac}}, k_{\text{cpa}}), \mathsf{ct})$: *Compute* $(m \,\|\, \sigma) \leftarrow \mathsf{Dec}_{\text{cpa}}(k_{\text{cpa}}, \mathsf{ct})$*. Output* m *if* $\mathsf{Verify}(k_{\text{mac}}, m, \sigma) = 1$.

$\Diamond$

This seems to address the vulnerabilities that were present in the previous 'Encrypt and MAC' candidate. First, since the signature is encrypted, even if σ reveals m, the final ciphertext does not reveal σ in the clear, and therefore does not reveal m. As expected, this candidate is CPA secure.

(*) **Exercise 5.10.** *Let $\mathcal{I} = (\mathsf{Sign}, \mathsf{Verify})$ be a secure MAC scheme, and $\mathcal{E}_{\mathrm{cpa}} = (\mathsf{Enc}_{\mathrm{cpa}}, \mathsf{Dec}_{\mathrm{cpa}})$ is a CPA secure encryption scheme. Prove that the MAC-then-encrypt combination of $\mathcal{I}$ and $\mathcal{E}_{\mathrm{cpa}}$ is a CPA secure encryption scheme.*

However, this construction does not satisfy ciphertext integrity. CPA security does not prevent malleability attacks. As a result, an adversary, given an encryption of m (which is an $\mathcal{E}_{\mathrm{cpa}}$ encryption of $(\sigma \| m)$), an adversary can produce a fresh $\mathcal{E}_{\mathrm{cpa}}$ encryption of $(\sigma \| m)$ and break ciphertext integrity.

The early versions of SSL/TLS used MAC-then-Encrypt, and they were subject to padding oracle attacks [Vau02]. The signature ensures integrity of the message, but not the padding added.

(*) **Exercise 5.11.** *The early versions of SSL/TLS used CBC mode of encryption with the following padding.*[a] *Given a message m with $16 - t$ bytes in the last block, it would add $t - 1$ arbitrary bytes, followed by the number 't'. During decryption, it would perform CBC-decryption and ignore the last t bytes. If $t = 0$, then it adds a new block with 15 arbitrary bytes, followed by the number '16'. Encryption using this mode is CPA secure (even with this lazy padding). Show that it does not satisfy ciphertext integrity or CCA security.*

[a] I call this 'lazy padding' because the encryptor is adding arbitrary bytes instead of adding zeroes.

Encrypt then MAC. Encrypting the message, and then computing a MAC on the ciphertext results in a secure authenticated encryption scheme.

Construction 5.37 (Encrypt-then-MAC:). *Given a MAC scheme $(\mathsf{Sign}, \mathsf{Verify})$ with key space $\mathcal{K}_{\mathrm{mac}}$ and a CPA secure encryption scheme $(\mathsf{Enc}_{\mathrm{cpa}}, \mathsf{Dec}_{\mathrm{cpa}})$ with key space $\mathcal{K}_{\mathrm{cpa}}$, the authenticated encryption scheme, with key space $\mathcal{K}_{\mathrm{mac}} \times \mathcal{K}_{\mathrm{cpa}}$ works as follows:*

- $\mathsf{Enc}((k_{\mathrm{mac}}, k_{\mathrm{cpa}}), m)$: *Compute* $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathrm{cpa}}(k_{\mathrm{cpa}}, m)$, *and* $\sigma \leftarrow \mathsf{Sign}(k_{\mathrm{cpa}}, \mathsf{ct})$. *Output* (ct, σ).
- $\mathsf{Dec}((k_{\mathrm{mac}}, k_{\mathrm{cpa}}), (\mathsf{ct}, \sigma))$: *Compute* $m \leftarrow \mathsf{Dec}_{\mathrm{cpa}}(k_{\mathrm{cpa}}, \mathsf{ct})$. *Output m if* $\mathsf{Verify}(k_{\mathrm{mac}}, \mathsf{ct}, \sigma) = 1$.

CPA security of this scheme follows from CPA security of $\mathcal{E}_{\mathrm{cpa}}$, while ciphertext integrity follows from the MAC security. ◊

(*) **Exercise 5.12.** *Let $\mathcal{I} = (\mathsf{Sign}, \mathsf{Verify})$ be a secure MAC scheme, and $\mathcal{E}_{\mathrm{cpa}} = (\mathsf{Enc}_{\mathrm{cpa}}, \mathsf{Dec}_{\mathrm{cpa}})$ is a CPA secure encryption scheme. Prove that the Encrypt-then-MAC combination of $\mathcal{I}$ and $\mathcal{E}_{\mathrm{cpa}}$ (Construction 5.37) is a secure authenticated encryption scheme (that is, prove that it is both CPA secure and satisfies ciphertext integrity).*

Lecture 17: October 10th, 2023

Some incorrect constructions/uses of authenticated encryption

TBD

Chapter Summary and References

6 Public Key Encryption

Lecture 18: October 13th, 2023

In the last section, we saw how to handle active attacks against symmetric key encryption schemes. From this section onwards, we will switch our attention towards public key cryptography, starting with key agreement protocols. We will need new cryptographic assumptions for building public key cryptography, since OWFs and PRFs are not sufficient. Instead, we will rely on hard number-theoretic computational problems. In this section, we will first define public key encryption (PKE), and discuss some generic properties of PKE. Next, we will discuss a key agreement protocol (that allows Alice and Bob to share a secret key, while an eavesdropper learns no information about the key). This key agreement protocol will lead us to our first PKE scheme. Finally, we will see two other PKE schemes, based on other number-theoretic problems.

6.1 *PKE: Definitions*

As the name suggests, a public key encryption scheme is one that allows parties to encrypt messages using a 'public key' that is known to everyone. However, decryption is possible only if one has the corresponding secret key. We start with the syntax for PKE.

Syntax. A public key encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ consists of three p.p.t. algorithms with the following syntax:

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm is a randomized algorithm that takes as input the security parameter, and outputs a public key pk and a secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, m)$: The encryption algorithm takes as input a public key pk, message m, and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: The decryption algorithm takes as input a secret key sk, ciphertext ct, and outputs a message m or $\perp$.

The definition of correctness is similar to what we had for the secret key setting : for every $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for every $m \in \mathcal{M}$,

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$$

Security Definition(s). The security definition(s) are also similar to the symmetric key setting. Here, the adversary first gets a public key (since the public key is known to everyone), and we require that even if the adversary sees many ciphertexts, it does not learn any information about the underlying messages. Note that we are working with passive adversaries for now. A natural adaptation of the SKE definition of security against Chosen Plaintext Attacks (Definition 4.12) gives us the following security game.

Similar to what we did in the SKE case, we can also add 'training queries', where the adversary sends a message m and receives $\mathsf{Enc}(\mathsf{pk}, m)$. However, such training queries are useless in the PKE setting, since the adversary has pk and can compute $\mathsf{Enc}(\mathsf{pk}, m)$ by itself.

In order to make this security game stronger, we can add decryption queries. This corresponds to chosen ciphertext attacks *in the public key setting; we will discuss this in a later section.*

Many-time Security Game for PKE

1. Challenger picks $b \leftarrow \{0,1\}$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$. It sends pk to the adversary.
2. The adversary makes polynomially many queries. For the i^{th} query, it sends two messages $(m_{i,0}, m_{i,1})$. The challenger computes $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{i,b})$ and sends ct_i.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 22: In this security game, the adversary receives the public key, then sends pairs of messages adaptively. The challenger picks a bit b at the start of the game, and for each query, it encrypts one of the two messages.

Let us consider the following relaxation of the security game, where the adversary receives only one query. Recall, in the case of SKE, one-time semantic security is not equivalent to many-time CPA security (Shannon's one-time pad satisfies one-time security, but is not many-time CPA secure). However, in the PKE setting, one-time security and many-time security are equivalent. Let us first define the security game formally.

One-time Security Game for PKE

1. Challenger picks $b \leftarrow \{0,1\}$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$. It sends pk to the adversary.
2. The adversary sends two messages (m_0, m_1). The challenger computes $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ and sends ct.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 23: In this security game, the adversary receives the public key, then sends one pair of messages. The challenger picks a bit b and encrypts one of the two messages.

Lemma 6.1. *If there exists a p.p.t. adversary $\mathcal{A}$ that succeeds in Game 22 with non-negligible advantage, then there exists a p.p.t. reduction $\mathcal{B}$ that succeeds in Game 23 with non-negligible advantage.*

The proof goes via a sequence of hybrid experiments. First, let us consider a failed attempt. Suppose we try to prove this lemma directly, without any hybrids. Let $\mathcal{A}$ be a p.p.t. adversary $\mathcal{A}$ that plays Game 22 with the challenger, and let World-b denote the scenario where challenger picks bit b. Suppose we are given that $p_b = \Pr[\mathcal{A} \text{ outputs } 0 \text{ in World-}b]$, and suppose $|p_0 - p_1|$ is non-negligible.

Consider the following natural reduction algorithm that uses $\mathcal{A}$ to win Game 23. The reduction algorithm picks a bit $\tilde{b}$. It receives the public key pk from the challenger, which it forwards to $\mathcal{A}$. Next, it receives queries from $\mathcal{A}$. For all but the last query, the reduction computes $\mathsf{ct}_i \leftarrow \mathsf{Enc}\left(\mathsf{pk}, m_{i,\tilde{b}}\right)$ and sends ct_i to the adversary. For the last query $(m_{0,q}, m_{1,q})$, it sends them to the challenger. It receives a ciphertext ct^*, which it forwards to the adversary. Finally, the adversary sends a bit b', which it forwards to the challenger. However, we cannot prove that the advantage of this reduction is $|p_0 - p_1|$ (why?).

Lecture 19: October 14th, 2023

Proof. Suppose the adversary makes q queries in Game 22. We will consider $q+1$ hybrid experiments, for $i \in [0, q]$.

Hybrid-World-i:

1. Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{pke}}(1^\lambda)$, sends pk to $\mathcal{A}$.
2. $\mathcal{A}$ sends polynomially many encryption queries. For the j^{th} query $m_{j,0}, m_{j,1}$, the challenger does the following:
   - If $j \leq i$, challenger sends $\mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{j,1})$.
   - If $j > i$, challenger sends $\mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{j,0})$.
3. $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},i}$$

Note that Hybrid-World-0 corresponds to World-0 and Hybrid-World-q corresponds to World-1. Since we are assuming that $\mathcal{A}$ wins Game 22, $|p_0 - p_1|$ is non-negligible. As a result, there exists an index $i \in [0, q]$ such that $|p_{\mathsf{Hyb},i} - p_{\mathsf{Hyb},i+1}|$ is non-negligible. The existence of such an index implies a single-query CPA attack on $\mathcal{E}$ as discussed in Observation 6.2 below.

Observation 6.2. *Suppose there exists a p.p.t. adversary $\mathcal{A}$ that makes q queries in Game 22, and $|p_{\mathsf{Hyb},i} - p_{\mathsf{Hyb},i+1}|$ is non-negligible. Then there exists a p.p.t. reduction $\mathcal{B}$ that wins Game 23 with non-negligible advantage.*

The reduction algorithm receives the public key pk from the Game 23 challenger, which it forwards to the adversary $\mathcal{A}$. Next, it receives q queries from $\mathcal{A}$. For the first i queries, the reduction computes $\mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{j,1})$. For the $(i+1)^{\text{th}}$ query, the reduction forwards $(m_{i+1,0}, m_{i+1,1})$ to the challenger, and forwards the challenge ciphertext ct^*. For the remaining queries, the reduction algorithm computes $\mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{j,0})$ and sends ct_j. Finally, the adversary sends its guess b', which the reduction forwards to the challenger. Check that the advantage of $\mathcal{B}$ is $|p_{\mathsf{Hyb},i} - p_{\mathsf{Hyb},i+1}|$.

□

Since the two security games are equivalent, we will use the simpler one to define *passive security* of public key encryption schemes.

Definition 6.3 (PKE: Security against chosen plaintext attacks)**.** *An encryption scheme $\mathcal{E}$ is said to be secure against chosen plaintext attacks (or simply CPA secure) if, for any p.p.t. adversary,*

$$\mathsf{CPAAdv}[\mathcal{A}, \mathcal{E}] = \left| \Pr[\mathcal{A} \text{ wins Game 23 w.r.t. } \mathcal{E}] - \frac{1}{2} \right|$$

is negligible. ◊

Encrypting long messages

Later in the section, we will see CPA secure encryption schemes with message space $\mathcal{M}_\lambda = \{0,1\}^\lambda$, where the encryption of a λ bit message will be a 2λ bit ciphertext. How can we handle public key encryption of long messages? Let $\mathcal{E}$ be a CPA secure PKE scheme with message space $\{0,1\}^\lambda$, and suppose we want to use $\mathcal{E}$ to encrypt unbounded length messages $\left(\text{that is, } \mathcal{M} = \left(\{0,1\}^\lambda\right)^{\geq 1}\right)$.

A natural idea is the following: for any message $m = (m_1, m_2, \ldots, m_\ell)$, output $\mathsf{ct} = (\mathsf{Enc}(\mathsf{pk}, m_1), \mathsf{Enc}(\mathsf{pk}, m_2), \ldots, \mathsf{Enc}(\mathsf{pk}, m_\ell))$. This is CPA secure. However, it is inefficient for two reasons: the size of the ciphertext is $2 \cdot \lambda \cdot \ell$ (recall, for SKE, we could achieve CPA security with ciphertext size being $\lambda + |m| = \lambda + \ell \cdot \lambda$ bits). Next, suppose the time required to encrypt each component is $T_{\text{pub}}(\lambda)$. Then the time required to compute the encryption of m is $\ell \cdot T_{\text{pub}}(\lambda)$. *Can we do better?*

Indeed, we can do better, and it is possible to achieve ciphertext size $2\lambda + \ell \cdot \lambda$ bits, and the idea is to appropriately combine a CPA-secure PKE scheme with a *one-time semantically secure* SKE scheme $\mathcal{E}_{\text{ske}} = (\mathsf{Enc}_{\text{ske}}, \mathsf{Dec}_{\text{ske}})$. Recall we can build such SKE schemes using a secure pseudorandom generator. This combination of PKE and SKE is called *hybrid encryption*.

Construction 6.4 (Hybrid Encryption)**.**
Let $\mathcal{E}_{\text{pke}} = (\mathsf{Setup}_{\text{pke}}, \mathsf{Enc}_{\text{pke}}, \mathsf{Dec}_{\text{pke}})$ be a CPA-secure public key encryption scheme with message space $\{0,1\}^\lambda$ and ciphertext space $\{0,1\}^{2\lambda}$. Let $\mathcal{E}_{\text{ske}} = (\mathsf{Enc}_{\text{ske}}, \mathsf{Dec}_{\text{ske}})$ a one-time semantically secure SKE scheme with key space $\{0,1\}^\lambda$, and message space $\{0,1\}^{\geq 1}$. Consider the following combination of $\mathcal{E}_{\text{pke}}$ and $\mathcal{E}_{\text{ske}}$:

- $\mathsf{Setup}(1^\lambda)$: *Compute* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\text{pke}}(1^\lambda)$. *Output* pk *as the public key and* sk *as the secret key.*
- $\mathsf{Enc}(\mathsf{pk}, m)$: *Choose a key* $k \leftarrow \{0,1\}^\lambda$, *compute* $\mathsf{ct}_1 \leftarrow \mathsf{Enc}_{\text{pke}}(\mathsf{pk}, k)$ *and* $\mathsf{ct}_2 \leftarrow \mathsf{Enc}_{\text{ske}}(k, m)$. *Output* $(\mathsf{ct}_1, \mathsf{ct}_2)$.
- $\mathsf{Dec}(\mathsf{sk}, (\mathsf{ct}_1, \mathsf{ct}_2))$: *Compute* $k \leftarrow \mathsf{Dec}_{\text{pke}}(\mathsf{sk}, \mathsf{ct}_1)$ *and output* $\mathsf{Dec}_{\text{ske}}(k, \mathsf{ct}_2)$.

If the SKE scheme is built using a secure PRG (eg. Construction 3.8), then the resulting PKE scheme is very efficient. The size of the ciphertext, corresponding to encryption of message m is $2\lambda + |m|$ bits. ◊

Claim 6.5. *Assuming $\mathcal{E}_{\text{pke}}$ is a CPA secure PKE scheme (Definition 6.3) and $\mathcal{E}_{\text{ske}}$ is a one-time semantically secure SKE scheme (Definition 3.2), the hybrid encryption scheme (Construction 6.4) is a CPA secure PKE scheme.*

Proof idea: The proof will proceed via a hybrid argument. We switch to a hybrid world where the challenger picks two SKE keys k, k'. It encrypts k using the PKE scheme, but encrypts the message using k'. This switch cannot be detected by the adversary (otherwise CPA security of the PKE scheme would be broken). Once we make this switch, we can use the semantic security of the SKE scheme.

Proof. We will consider two hybrid worlds for this proof. First, let us recall World-0 and World-1.

World-0:

1. Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathrm{pke}}(1^\lambda)$, sends pk to $\mathcal{A}$.
2. $\mathcal{A}$ sends m_0, m_1.
3. Chall. chooses $k \leftarrow \{0,1\}^\lambda$. It sends $\mathsf{ct}_1 \leftarrow \mathsf{Enc}_{\mathrm{pke}}(\mathsf{pk}, k)$ and $\mathsf{ct}_2 \leftarrow \mathsf{Enc}_{\mathrm{ske}}(k, m_0)$.
4. $\mathcal{A}$ outputs b'.

$\Pr[\mathcal{A} \text{ outputs } 0] = p_0$

World-1:

1. Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathrm{pke}}(1^\lambda)$, sends pk to $\mathcal{A}$.
2. $\mathcal{A}$ sends m_0, m_1.
3. Chall. chooses $k \leftarrow \{0,1\}^\lambda$. It sends $\mathsf{ct}_1 \leftarrow \mathsf{Enc}_{\mathrm{pke}}(\mathsf{pk}, k)$ and $\mathsf{ct}_2 \leftarrow \mathsf{Enc}_{\mathrm{ske}}(k, m_1)$.
4. $\mathcal{A}$ outputs b'.

$\Pr[\mathcal{A} \text{ outputs } 0] = p_1$

Next, we will define the two hybrid worlds: hybrid-world-0, and hybrid-world-1. Hybrid-world-0 (resp. hybrid-world-1) is exactly identical to world-0 (resp. world-1), except that the challenger picks two keys k, k', encrypts k' using the public key, and uses k to SKE-encrypt m_0 (resp. m_1).

Hybrid-World-0:

1. Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathrm{pke}}(1^\lambda)$, sends pk to $\mathcal{A}$.
2. $\mathcal{A}$ sends m_0, m_1.
3. Chall. chooses $k, k' \leftarrow \{0,1\}^\lambda$. It sends $\mathsf{ct}_1 \leftarrow \mathsf{Enc}_{\mathrm{pke}}(\mathsf{pk}, k')$ and $\mathsf{ct}_2 \leftarrow \mathsf{Enc}_{\mathrm{ske}}(k, m_0)$.
4. $\mathcal{A}$ outputs b'.

$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},0}$

Hybrid-World-1:

1. Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathrm{pke}}(1^\lambda)$, sends pk to $\mathcal{A}$.
2. $\mathcal{A}$ sends m_0, m_1.
3. Chall. chooses $k, k' \leftarrow \{0,1\}^\lambda$. It sends $\mathsf{ct}_1 \leftarrow \mathsf{Enc}_{\mathrm{pke}}(\mathsf{pk}, k')$ and $\mathsf{ct}_2 \leftarrow \mathsf{Enc}_{\mathrm{ske}}(k, m_1)$.
4. $\mathcal{A}$ outputs b'.

$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},1}$

Using the CPA security of $\mathcal{E}_{\mathrm{pke}}$, we can argue that p_b and $p_{\mathsf{Hyb},b}$ are negligibly close, and using the one-time semantic security of $\mathcal{E}_{\mathrm{ske}}$, we can conclude that $p_{\mathsf{Hyb},0}$ and $p_{\mathsf{Hyb},1}$ are negligibly close. These are summarized in the observation below. Both these observations require a reduction-based proof.

Observation 6.6. *Assuming $\mathcal{E}_{\mathrm{pke}}$ is a CPA secure PKE scheme, for any p.p.t. adversary $\mathcal{A}$ and bit $b \in \{0,1\}$, $|p_b - p_{\mathsf{Hyb},b}| \leq \mathrm{negl}(\lambda)$.*

Observation 6.7. *Assuming $\mathcal{E}_{\mathrm{ske}}$ is a one-time semantically secure SKE scheme, for any p.p.t. adversary $\mathcal{A}$, $|p_{\mathsf{Hyb},0} - p_{\mathsf{Hyb},1}| \leq \mathrm{negl}(\lambda)$.*

Putting these two together, we can conclude that p_0 is negligibly close to p_1. □

Now that we have seen the definition(s) and some generic properties of public key encryption, let us see how to build a PKE scheme using number-theoretic computational problems. We will start with a simpler problem: Alice and Bob want to establish a secret key in the presence of an eavesdropping adversary. Note that this is simpler than PKE. If we had a PKE scheme, Alice could simply choose a public/secret key and send the public key to Bob. Next, Bob can pick a random secret key, encrypt it using Alice's public key and send the encryption to Alice. Alice can then recover the key using her secret key.

Whitfield Diffie and Martin Hellman gave the first key-agreement protocol [DH76]. Their work also laid the foundations of public key cryptography, and they were awarded the Turing award in 2015 for their contributions to modern cryptography.

6.2 *Diffie-Hellman's Key Exchange Protocol*

Brief Intro to Prime Order Groups

A (commutative) group is a set $\mathbb{G}$ together with an operation '$\cdot$' defined on the elements of $\mathbb{G}$, with the following properties:

- (*identity*) $\exists 1_{\mathbb{G}} \in \mathbb{G}$ such that for all $x \in \mathbb{G}$, $1_{\mathbb{G}} \cdot x = x \cdot 1_{\mathbb{G}} = x$
- (*closure*) for all $x, y \in \mathbb{G}$, $x \cdot y \in \mathbb{G}$
- (*commutativity*) for all $x, y \in \mathbb{G}$, $x \cdot y = y \cdot x$
- (*associativity*) for all $x, y, z \in \mathbb{G}$, $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- (*inverse*) for all $x \in \mathbb{G}$, there exists a $y \in \mathbb{G}$ such that $x \cdot y = 1_{\mathbb{G}}$

The order of the group is $|\mathbb{G}|$. We will work with prime order groups. Let q denote the order of the group. Since we are interested in computational properties, we require the following additional properties:

- (*efficient representation*) every group element can be efficiently represented. That is, there exists a polynomial $p_1(\cdot)$ such that every group element can be represented using $p_1(\log q)$ bits.
- (*efficient group operation*) it is possible to efficiently compute the group operation. That is, there exists a polynomial $p_2(\cdot)$ and an algorithm such that given a representation of $g_1, g_2 \in \mathbb{G}$ as input, it outputs a representation of $g_1 \cdot g_2$ in $p_2(\log q)$ steps.
- (*efficient samplability*) there exists a polynomial $p_3(\cdot)$ and an algorithm that takes no input, and samples a uniformly random group element of $\mathbb{G}$ in time $p_3(\log q)$.

In fact, we require an infinite family of such groups $\{\mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}}$, one for each security parameter. Let $\mathcal{G}$ be an efficient algorithm that takes as input a security parameter λ (in unary), and outputs a prime $q = \Theta(2^\lambda)$, together with a random group element $g \leftarrow \mathbb{G}_\lambda$ and a description of the group operation '$\cdot$'. Additionally, we assume that for every λ, the algorithm $\mathcal{G}$ outputs the same q, same $g \in \mathbb{G}_\lambda$ and group operation '$\cdot$'. We will use $\mathcal{G}$ to refer to the infinite family of groups.

Examples of prime-order groups:

- Let q be a prime. The set $\mathbb{Z}_q$, together with the operation 'addition mod q', form a group.
- Let q, p be primes such that $p = 2 \cdot q + 1$. Let g be an element in $\mathbb{Z}_p^*$ such that $g^q = 1 \bmod p$, and $g \neq 1$. Let $\mathbb{G} = \{g^i \bmod p : i \in [0, q-1]\}$, and the group operation is 'multiplication modulo p'.

In a prime-order group, any non-identity element is a generator of the group. That is, consider any $g \in \mathbb{G}$, $g \neq 1_{\mathbb{G}}$. Then $\{ g^i : i \in \mathbb{Z}_q \} = \mathbb{G}$. As a result, there is a natural bijective mapping between $(\mathbb{G}, \cdot)$ and $(\mathbb{Z}_q, \text{addition modulo } q)$ that is preserved under the respective group operations. This mapping $\phi : \mathbb{G} \to \mathbb{Z}_q$ maps g^i to i, and note that $\phi(g^i \cdot g^j) = (i + j) \bmod q$. Such bijective mappings between two groups are called *isomorphisms*.
For a mathematician, all prime-order groups are the same (due to the isomorphism between $\mathbb{G}$ and $\mathbb{Z}_q$). However, for a computer scientist, these groups could behave differently. For instance, certain computational problems are easy on $(\mathbb{Z}_q, \text{addition mod } q)$ but potentially hard on some other q-order group.

The starting point for Diffie-Hellman's protocol is the observation that the discrete log problem is hard for certain prime-order groups.

Definition 6.8. *Let $\mathcal{G}$ be an infinite family of groups. We say that the discrete log problem is hard for $\mathcal{G}$ if, for any p.p.t. adversary $\mathcal{A}$, the following probability is negligible:*

$$\Pr\left[\begin{array}{c} (q, g, \cdot) \leftarrow \mathcal{G}(1^\lambda), a \leftarrow \mathbb{Z}_q, \\ a \leftarrow \mathcal{A}(q, g, g^a) \end{array} \right].$$

◊

The discrete log problem is not hard for many prime-order groups. For instance, let q be a prime, consider the set $\mathbb{Z}_q$ with group operation 'addition modulo q'. Given q, a random number $g \in \mathbb{Z}_q$ and $a \times g \bmod q$ for a random $a \leftarrow \mathbb{Z}_q$, one can easily compute a (how?).

An important point that came up in class: all prime-order groups are isomorphic to $\mathbb{Z}_q$, then why is discrete log easy on some groups, and potentially hard on others? The reason is that the isomorphism between these groups may not be efficiently computable.

Diffie and Hellman used a discrete-log-hard group family $\mathcal{G}$ for their key exchange protocol.

Construction 6.9 (Diffie-Hellman Key Exchange Protocol)**.**
Let $\mathcal{G}$ be a group family such that the discrete log problem is hard for $\mathcal{G}$. Alice and Bob use $\mathcal{G}$ for deriving a shared key as follows.

1. *Alice chooses $(q, g, \cdot) \leftarrow \mathcal{G}(1^\lambda)$, $a \leftarrow \mathbb{Z}_q$ and sends $(q, g, A = g^a)$ to Bob.*
2. *Bob chooses $b \leftarrow \mathbb{Z}_q$ and sends $B = g^b$.*
3. *At this point, both Alice and Bob can efficiently compute $g^{a \cdot b}$. Alice can compute B^a, while Bob can compute A^b.*

The adversary can see all the communication between Alice and Bob. In particular, it sees (q, g, g^a, g^b). However, it should not learn $g^{a \cdot b}$ given just g, g^a, g^b. ◊

Diffie and Hellman conjectured that for certain groups where the discrete log problem is hard, the key derived by Alice and Bob at the end of the protocol is indistinguishable from a uniformly random group element. We call such groups *DDH-hard groups* (DDH refers to *Decisional Diffie-Hellman*).

Definition 6.10. *Let $\mathcal{G}$ be an infinite family of groups. We say that the Decisional Diffie-Hellman problem is hard for $\mathcal{G}$ if, for any p.p.t. adversary $\mathcal{A}$, the following quantity is negligible:*

$$\left|\mathsf{Pr}\left[\begin{array}{c}(q,g,\cdot)\leftarrow\mathcal{G}(1^\lambda),a,b\leftarrow\mathbb{Z}_q,\\ 0\leftarrow\mathcal{A}\left(q,g,g^a,g^b,g^{a\cdot b}\right)\end{array}\right]-\mathsf{Pr}\left[\begin{array}{c}(q,g,\cdot)\leftarrow\mathcal{G}(1^\lambda),a,b,c\leftarrow\mathbb{Z}_q,\\ 0\leftarrow\mathcal{A}\left(q,g,g^a,g^b,g^c\right)\end{array}\right]\right|.$$

◊

In other words, no efficient adversary can distinguish between the following distributions:

$$\mathcal{D}_0=\left\{\left(q,g,g^a,g^b,g^{a\cdot b}\right):\ \ (q,g,\cdot)\leftarrow\mathcal{G}(1^\lambda),a,b\leftarrow\mathbb{Z}_q\ \right\}$$
$$\mathcal{D}_1=\left\{\left(q,g,g^a,g^b,g^c\right):\ \ (q,g,\cdot)\leftarrow\mathcal{G}(1^\lambda),a,b,c\leftarrow\mathbb{Z}_q\ \right\}$$

This is a 'decisional' problem since the adversary must distinguish between two distributions. We can also consider a 'computational' problem where the adversary, given (q,g,g^a,g^b), must compute $g^{a\cdot b}$. This is the *Computational Diffie-Hellman* (CDH) problem. Note that, for any family of groups, the discrete log problem is at least as hard as the CDH problem, which in turn is at least as hard as the DDH problem.

Lecture 20: October 17th, 2023

One can also consider other group-theoretic computational problems. For instance, consider the following problem: given (q,g,g^a) for a random $a\leftarrow\mathbb{Z}_q$, compute g^{a^2}. Suppose an algorithm $\mathcal{A}$ could efficiently compute g^{a^2} given (g,g^a). Then there exists a reduction algorithm that solves CDH (and hence, also DDH). The reduction algorithm, on receiving $\left(g,A=g^a,B=g^b\right)$ from the challenger, sends $(g,A\cdot B)$ to $\mathcal{A}$, and receives $C_1=g^{(a+b)^2}$. Next, it sends $\left(g,A\cdot B^{-1}\right)$, and receives $C_2=g^{(a-b)^2}$. It computes $C=C_1\cdot C_2^{-1}=g^{4\cdot a\cdot b}$. Finally, it needs to compute the fourth-root of C. Let z be the unique integer in $\mathbb{Z}_q$ such that $4\cdot z=1 \bmod q$. The reduction outputs C^z (which can be computed efficiently, why?).

6.3 *A PKE scheme based on DDH*

Tahir Elgamal, in 1985, gave the following scheme for public key encryption, which can be proven CPA secure based on the Decisional Diffie-Hellman problem.

Construction 6.11 (Elgamal's Public Key Encryption Scheme)**.** *Let $\mathcal{G}$ be a DDH-hard family of groups. The message space, for security parameter λ, is the group $\mathbb{G}_\lambda$.*

- $\mathsf{Setup}(1^\lambda)$: *The setup algorithm first generates the group order, the group generator and group operation $(q,g,\cdot)\leftarrow\mathcal{G}(1^\lambda)$. Next, it samples $a\leftarrow\mathbb{Z}_q$, sets $\mathsf{pk}=(q,g,A=g^a)$ and $\mathsf{sk}=a$.*
- $\mathsf{Enc}(\mathsf{pk}=(q,g,A),m\in\mathbb{G}_\lambda)$: *The encryption algorithm chooses $r\leftarrow\mathbb{Z}_q$ and sets $\mathsf{ct}=(g^r,A^r\cdot m)$.*
- $\mathsf{Dec}(\mathsf{sk},\mathsf{ct}=(\mathsf{ct}_1,\mathsf{ct}_2))$: *The decryption algorithm outputs $\mathsf{ct}_2\cdot\left(\mathsf{ct}_1^{\mathsf{sk}}\right)^{-1}$.*

◊

Security of Elgamal's scheme follows from the DDH assumption.

Claim 6.12. *Assuming the DDH problem is hard for $\mathcal{G}$, Construction 6.11 is a CPA secure public key encryption scheme.*

Proof. The proof follows via a sequence of hybrids.

- World-0: Challenger sends $\mathsf{pk} = (g, g^a)$ to the adversary. The adversary sends m_0, m_1, and receives $(g^r, m_0 \cdot g^{a \cdot r})$. Finally, the adversary outputs bit b', and let p_0 denote the probability of adversary outputting 0.
- Hyb-World-0: Challenger sends $\mathsf{pk} = (g, g^a)$ to the adversary. The adversary sends m_0, m_1, and receives $(g^r, m_0 \cdot g^c)$. Finally, the adversary outputs bit b', and let $p_{\mathsf{Hyb},0}$ denote the probability of adversary outputting 0.
- Hyb-World-1: Challenger sends $\mathsf{pk} = (g, g^a)$ to the adversary. The adversary sends m_0, m_1, and receives $(g^r, m_1 \cdot g^c)$. Finally, the adversary outputs bit b', and let $p_{\mathsf{Hyb},1}$ denote the probability of adversary outputting 0.
- World-1: Challenger sends $\mathsf{pk} = (g, g^a)$ to the adversary. The adversary sends m_0, m_1, and receives $(g^r, m_1 \cdot g^{a \cdot r})$. Finally, the adversary outputs bit b', and let p_1 denote the probability of adversary outputting 0.

If $\left|p_\beta - p_{\mathsf{Hyb},\beta}\right|$ is non-negligible, then there exists a p.p.t. algorithm that breaks the DDH assumption. The reduction algorithm receives (g, A, B, T) from the challenger, where $A = g^a$, $B = g^b$ and T is either $g^{a \cdot b}$ or g^c. It sends $\mathsf{pk} = (g, A)$ to the adversary. On receiving m_0, m_1 from the adversary, the reduction sends $(B, m_\beta \cdot T)$ to the adversary. Finally, it receives a bit from the adversary, which it forwards to the challenger. If $T = g^{a \cdot b}$, then the adversary is participating in World-β (where the probability of outputting 0 is p_β), else it is participating in Hyb-World-β (where the probability of outputting 0 is $p_{\mathsf{Hyb},\beta}$). Therefore, the reduction algorithm's advantage in the DDH game is $\left|p_\beta - p_{\mathsf{Hyb},\beta}\right|$.

Finally, note that $p_{\mathsf{Hyb},0} = p_{\mathsf{Hyb},1}$. This is because g^c is a uniformly random element in $\mathbb{G}$, and therefore $m_0 \cdot g^c$ is perfectly indistinguishable from $m_1 \cdot g^c$. □

Anonymous PKE

CPA security ensures that an adversary, given the ciphertext, does not learn anything about the underlying message. However, the Elgamal encryption scheme ensures that the ciphertext also hides the public key used for encrypting the message. Encryption schemes satisfying this security are called *anonymous*, and public key anonymity is captured via the following security game (Figure 24).

Definition 6.13. *A public key encryption scheme is said to be an anonymous scheme if for all p.p.t. adversaries $\mathcal{A}$, the advantage in the anonymity game (described in Figure 24) is negligible.* ◊

Elgamal's scheme (Construction 6.11) is both CPA secure and anonymous. The anonymity proof is very similar to the proof of Claim 6.12 (follows the same sequence of hybrids).

Anonymity of public key for PKE

1. Challenger picks $b \leftarrow \{0,1\}$ and $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Setup}\left(1^\lambda\right)$, $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Setup}\left(1^\lambda\right)$. It sends $\mathsf{pk}_0, \mathsf{pk}_1$ to the adversary.
2. The adversary sends a message m. The challenger computes $\mathsf{ct} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_b, m\right)$ and sends ct.
3. Finally, the adversary outputs its guess b', and wins the game if $b = b'$.

Figure 24: In this security game, the adversary receives two public keys, then sends a message m. The challenger picks a bit b, it encrypts m using one of the two public keys.

Active Attacks against Elgamal Scheme

The Elgamal scheme is susceptible to various active attacks.

- Given an encryption $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ of message $m \in \mathbb{G}$, an adversary can produce an encryption of $m \cdot m'$ by outputting $\mathsf{ct}' = (\mathsf{ct}_1, m' \cdot \mathsf{ct}_2)$.
- Given an encryption $\mathsf{ct}_1 = (\mathsf{ct}_{1,1}, \mathsf{ct}_{1,2})$ of m_1 and an encryption $\mathsf{ct}_2 = (\mathsf{ct}_{2,1}, \mathsf{ct}_{2,2})$ of m_2, an adversary can produce an encryption of $m_1^{c_1} \cdot m_2^{c_2}$ for any $c_1, c_2 \in \mathbb{Z}_q$. Note that $\left(\left(\mathsf{ct}_{1,1}\right)^{c_1} \cdot \left(\mathsf{ct}_{2,1}\right)^{c_2}, \left(\mathsf{ct}_{1,2}\right)^{c_1} \cdot \left(\mathsf{ct}_{2,2}\right)^{c_2}\right)$ is an encryption of $m_1^{c_1} \cdot m_2^{c_2}$.

The second 'vulnerability' can also be a useful feature when we want to perform linear computations on encrypted data. Below, we discuss one such application — private information retrieval.

Private Information Retrieval (PIR). Consider a service provider like Netflix that has n movies, each m bits long. Users can send an index $i \in [n]$, and receive the i^{th} movie in response. However, the service provider learns the indices queried by the user. Is it possible for a user to retrieve information from the database privately? A naive approach is the following: for each query, the service provider sends the entire database (resulting in $n \cdot m$ bits of communication). Can we do better than $n \cdot m$ bits of communication, while ensuring privacy of queries?

More formally, we want an algorithm Encode that takes as input a security parameter, and an index $i \in [n]$, and outputs an encoding enc and a secret state st (which will be used during decoding). The service provider takes as input a database DB, an encoding enc, and outputs a new encoding enc', which is sent to the user. Finally, the user uses enc' together with its secret state st to recover the i^{th} entry of the database. For privacy, we require that $\mathsf{Encode}\left(1^\lambda, i\right)$ is computationally indistinguishable from $\mathsf{Encode}\left(1^\lambda, j\right)$. Using Elgamal's encryption scheme, we can construct a PIR scheme where the total communication is $2(n+m)$ group elements (and hence $O((n+m)\,\mathrm{poly}(\lambda))$ bits).

Lecture 21: October 20th, 2023

Construction 6.14 (A PIR scheme based on Elgamal's scheme)**.**

- $\mathsf{Encode}\left(1^\lambda, i\right)$: *The encoding algorithm first chooses a public/secret key for Elgamal's scheme. It chooses* $a \leftarrow \mathbb{Z}_q$*. Next, it chooses* $r_j \leftarrow \mathbb{Z}_q$ *for each* $j \in [n]$*, sets* $\mathsf{ct}_{j,1} = g^{r_j}$*. The component* $\mathsf{ct}_{j,2}$ *is computed as follows:*

$$\mathsf{ct}_{j,2} = \begin{cases} g^{a \cdot r_j + 1} \text{ if } j = i \\ g^{a \cdot r_j} \text{ if } j \neq i \end{cases}$$

  Essentially, $\mathsf{ct}_j = (\mathsf{ct}_{j,1}, \mathsf{ct}_{j,2})$ *is an Elgamal encryption of* $g^1 = g$ *if* $i = j$*, else it is an encryption of* $g^0 = 1$*. The secret state is* $\mathsf{st} = a$*, and the encoding is* $\left((\mathsf{ct}_{j,1}, \mathsf{ct}_{j,2})\right)_{j \in [n]}$.

- $\mathsf{Process}\left(\mathsf{DB}, \mathsf{enc} = \left((\mathsf{ct}_{j,1}, \mathsf{ct}_{j,2})\right)_{j \in [n]}\right)$: *For* $k = 1$ *to* m*, the processing algorithm does the following:*

  - *it computes* $\mathsf{ct}'_{k,1} = \prod_{j=1}^{n} \mathsf{ct}_{j,1}^{\mathsf{DB}[j,k]}$ *and* $\mathsf{ct}'_{k,2} = \prod_{j=1}^{n} \mathsf{ct}_{j,2}^{\mathsf{DB}[j,k]}$.

  The processing algorithm outputs $\left(\left(\mathsf{ct}'_{k,1}, \mathsf{ct}'_{k,2}\right)\right)_{k \in [m]}$*. If* enc *is an encoding of* $i \in [n]$*, then note that* ct'_k *is an Elgamal encryption of* $g^{\mathsf{DB}[i,k]}$.

- $\mathsf{Decode}\left(\mathsf{st}, \left(\left(\mathsf{ct}'_{k,1}, \mathsf{ct}'_{k,2}\right)\right)\right)$: *For each* $k \in [m]$*, the decoding algorithm checks if* $\mathsf{ct}'_{k,2} = \left(\mathsf{ct}'_{k,1}\right)^a$*. If so, it sets the* k^{th} *bit to be* 0*, else it sets it to* 1.

$\Diamond$

6.4 *PKE and Trapdoor Functions*

The early attempts towards building public key encryption followed a different paradigm: they first built a stronger primitive called *trapdoor functions*, and then gave 'heuristic' constructions of PKE using TDFs. Some of the practically interesting schemes follow this approach. There are two other reasons for studying this approach: (a) we will see the RSA computational assumption, which is one of the most well-studied cryptographic assumptions (b) we will discuss how to prove security of 'heuristic' constructions in idealized security models.

Trapdoor Functions

A trapdoor function (TDF) is similar to a public key encryption scheme. It allows efficient evaluation on any input using a public key, and efficient inversion using the corresponding secret key. However, without the secret key, it is hard to recover the input. Formally, let $\mathcal{X}$ be the domain, and $\mathcal{Y}$ the co-domain. A trapdoor function family has three algorithms : setup, evaluation and inversion with the following syntax.

- $\mathsf{Setup}\left(1^\lambda\right)$: The setup algorithm outputs a public key pk and a secret key sk.

- $\mathsf{Eval}(\mathsf{pk}, x)$: The evaluation algorithm takes as input a public key pk, input $x \in \mathcal{X}$, and outputs $y \in \mathcal{Y}$.
- $\mathsf{Inv}(\mathsf{sk}, y)$: The inversion algorithm takes as input a secret key sk, $y \in \mathcal{Y}$, and outputs $x \in \mathcal{X}$ or $\perp$.

Correctness: For any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}$, $x \in \mathcal{X}$, $\mathsf{Inv}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, x)) = x$. Due to the correctness requirement, for every public key pk, the evaluation mapping is injective.

Security: Given the public key and evaluation on a random input, it is hard to recover the input.

Definition 6.15. *A tuple of algorithms* $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Inv})$ *is a secure trapdoor function family if, for any p.p.t. adversary* $\mathcal{A}$*, the following probability is negligible:*

$$\Pr\left[\begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}\left(1^\lambda\right) \\ x \leftarrow \mathcal{X}, y = \mathsf{Eval}(\mathsf{pk}, x) \\ \mathcal{A}(\mathsf{pk}, y) \text{ outputs } x\end{array}\right]$$

◊

While trapdoor functions are similar to PKE in spirit, a trapdoor function does not immediately yield a CPA secure PKE scheme. The most natural construction, where we set $\mathsf{Setup}_{\mathsf{pke}} = \mathsf{Setup}_{\mathsf{tdf}}$, $\mathsf{Enc}_{\mathsf{pke}}(\mathsf{pk}, m) = \mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, m)$ and $\mathsf{Dec}_{\mathsf{pke}}(\mathsf{sk}, \mathsf{ct}) = \mathsf{Inv}_{\mathsf{tdf}}(\mathsf{sk}, \mathsf{ct})$ is not CPA secure since encryption is deterministic. Let us consider the following candidates proposed in class.

Construction 6.16 (PKE from TDFs: Candidate 1)**.** *The input space of the TDF is* $\mathcal{X}$*, which we assume is equal to* $\{0,1\}^n$*. The message space of our encryption scheme is* $\mathcal{X}$*.*

- $\mathsf{Setup}_{\mathsf{pke}} = \mathsf{Setup}_{\mathsf{tdf}}$
- $\mathsf{Enc}_{\mathsf{pke}}(\mathsf{pk}, m)$: *Choose* $r \leftarrow \mathcal{X}$*, output* $(\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), r \oplus m)$.
- $\mathsf{Dec}_{\mathsf{pke}}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2))$: *Compute* $r = \mathsf{Inv}_{\mathsf{tdf}}(\mathsf{sk}, \mathsf{ct}_1)$*, output* $r \oplus \mathsf{ct}_2$.

This scheme is not CPA secure. The adversary, on receiving the public key pk*, sends* m_0, m_1 *and receives* $(\mathsf{ct}_1, \mathsf{ct}_2)$*. It checks if* $\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, \mathsf{ct}_2 \oplus m_0) = \mathsf{ct}_1$*. If so, it guesses that* m_0 *was encrypted, else it guesses* m_1 *was encrypted.* ◊

Construction 6.17 (PKE from TDFs: Candidate 2)**.**

- $\mathsf{Setup}_{\mathsf{pke}}\left(1^\lambda\right)$: *Sample* $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}\left(1^\lambda\right)$, $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}\left(1^\lambda\right)$*. The public key is* $(\mathsf{pk}_1, \mathsf{pk}_2)$*, and the secret key is* $(\mathsf{sk}_1, \mathsf{sk}_2)$.
- $\mathsf{Enc}_{\mathsf{pke}}(\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2), m)$: *Choose* $r \leftarrow \mathcal{X}$*, output* $(\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}_1, r), \mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}_2, r \oplus m))$.
- $\mathsf{Dec}_{\mathsf{pke}}(\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2), \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2))$: *Compute* $r = \mathsf{Inv}_{\mathsf{tdf}}(\mathsf{sk}_1, \mathsf{ct}_1)$*,* $z = \mathsf{Inv}_{\mathsf{tdf}}(\mathsf{sk}_2, \mathsf{ct}_2)$*, output* $z \oplus r$.

This scheme seems to address the vulnerability from Construction 6.16. However its CPA security cannot be proven based on the security of TDFs. Consider a contrived TDF family which reveals the last bit of the input. Such a function family can be a secure TDF, since the adversary needs to output the entire input. For such contrived TDFs, the above construction won't give a CPA secure PKE scheme. ◊

Proving security of heuristic constructions in idealized models

The initial PKE constructions based on TDFs were heuristic constructions. These constructions use a 'complicated' publicly-known, deterministic hash function $H : \{0,1\}^* \to \{0,1\}$.

Construction 6.18 (PKE from TDFs: Heuristic Construction)**.** *Let* $H : \{0,1\}^* \to \{0,1\}^\lambda$ *be a publicly-known deterministic function.*

- $\mathsf{Setup}_{\mathrm{pke}}\left(1^\lambda\right)$: *Sample* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathrm{tdf}}\left(1^\lambda\right)$. *The public key is* pk*, and the secret key is* sk.
- $\mathsf{Enc}_{\mathrm{pke}}(\mathsf{pk}, m)$: *Choose* $r \leftarrow \mathcal{X}$*, output* $(\mathsf{Eval}_{\mathrm{tdf}}(\mathsf{pk}, r), H(r) \oplus m)$.
- $\mathsf{Dec}_{\mathrm{pke}}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2))$: *Compute* $r = \mathsf{Inv}_{\mathrm{tdf}}(\mathsf{sk}, \mathsf{ct}_1)$*, output* $\mathsf{ct}_2 \oplus H(r)$.

◊

It is not clear how to prove security of the above construction in the standard model. Even if we assume H is one-way and collision-resistant, it does not suffice for proving security of Construction 6.18. However, this (and similar) constructions are widely used in practice. Can we get some security guarantees for such heuristic constructions?

The Random Oracle Model. The random oracle model is a well-studied security model for proving security of heuristic constructions. In this model, we assume the adversary does not get to see the internal workings of the (publicly-known) function H. Instead, it is allowed to query the function on any input of its choice. The challenger picks a uniformly random function at the start of the game, and uses this for answering all the queries.

Let us consider the security of Construction 6.18 in the random oracle model. At the start of the game, the challenger picks a uniformly random function H, as well as $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathrm{tdf}}\left(1^\lambda\right)$. It sends pk to the adversary. Next, the adversary is allowed to query the function H on any input x of its choice, and receives $H(x)$. It is allowed polynomially many queries, after which it sends m_0, m_1. The challenger picks a bit b, a random r and sends $(\mathsf{Eval}_{\mathrm{tdf}}(\mathsf{pk}, r), H(r) \oplus m_b)$ (note that the function H is used by the challenger for computing the challenge ciphertext). The adversary is then allowed more post-challenge oracle queries. Finally, it must output its guess.

Why is the random oracle model used for proving security? Isn't it too unrealistic,

since we are modeling a fixed, publicly-known, deterministic function using a 'random oracle'? Indeed, these idealized models are somewhat controversial. A security proof in the random oracle model does not give any formal security guarantees in the real-world. However, these give us some security guarantee for real-world heuristics. As an example, consider the following construction of PKE from TDFs (Construction 6.19).

Construction 6.19 (PKE from TDFs: Broken Heuristic Construction)**.** *Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a publicly-known deterministic function.*

- $\mathsf{Setup}_{\mathsf{pke}}\left(1^\lambda\right)$: *Sample* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}\left(1^\lambda\right)$. *The public key is* pk, *and the secret key is* sk.
- $\mathsf{Enc}_{\mathsf{pke}}(\mathsf{pk}, m)$: *Choose* $r \leftarrow \mathcal{X}$, *output* $(\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r),\ H(r) \oplus m,\ H(m))$.
- $\mathsf{Dec}_{\mathsf{pke}}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3))$: *Compute* $r = \mathsf{Inv}_{\mathsf{tdf}}(\mathsf{sk}, \mathsf{ct}_1)$, *output* $\mathsf{ct}_2 \oplus H(r)$.

This construction is not CPA secure, due to the last ciphertext component. ◊

This construction is broken, and an adversary can win the security game in the random oracle model. It sends two challenge messages m_0, m_1 and receives ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$. Next, it queries the random oracle on input m_0, m_1. One of them will be equal to ct_3, and this reveals whether m_0 or m_1 was encrypted. Note that this is a *generic attack*. The adversary does not require the structure of H, it only needs the input/output behaviour of H (which is provided by the random oracle). Hence, a security proof in the random oracle model tells us that there are no generic attacks against the scheme in the real-world. One can also view security proofs in the random oracle model as a sanity check for heuristic constructions.

Lecture 22: October 27th, 2023

(*) Exercise 6.1. *Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a publicly known hash function. Construct a MAC scheme using H, and prove security of your scheme in the random oracle model.*

Security proof of Construction 6.18 in the random oracle model.

Proof idea: Consider a slightly simplified random oracle model - the adversary receives a public key, then it must send two challenge messages m_0, m_1. It receives the encryption of one of these, and after receiving the challenge ciphertext, the adversary is allowed to query the random oracle on polynomially many inputs (that is, there are no pre-challenge random oracle queries, but post-challenge random oracle queries are allowed). Finally, it sends its guess.

Suppose the trapdoor function is completely insecure. Then the scheme will not be secure in the random oracle model. The adversary, on receiving $(\mathsf{ct}_1 = \mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), \mathsf{ct}_2 = H(r) \oplus m_b)$, will first recover r from ct_1. Next, it queries the random oracle on r, and then recovers m_b from ct_2.

However, if the TDF is secure, then the adversary should not be able to query on input r. And if it does not query on input r, then the adversary receives no

information about string $z = H(r)$, and therfore the adversary learns nothing about m_b from ct_2.

Claim 6.20. *Assuming* $(\mathsf{Setup}_{\mathsf{tdf}}, \mathsf{Eval}_{\mathsf{tdf}}, \mathsf{Inv}_{\mathsf{tdf}})$ *is a secure trapdoor function (Definition 6.15), Construction 6.18 is CPA secure in the random oracle model.*

Proof. The formal proof proceeds via a sequence of hybrids, gradually transitioning from World-0 to World-1. Note that since we are working in the random oracle model, the challenger picks a uniformly random function H, and the adversary is allowed queries to H in both World-0 and World-1 (and also in all the intermediate hybrids). Instead of choosing H at the start of the game, the challenger defines H 'on-the-fly' by picking uniformly random strings for each fresh query.

World-0:

1. (Setup) Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}(1^\lambda)$, sends pk to $\mathcal{A}$. It also maintains a table $\mathcal{T} = \varnothing$ for random oracle queries.
2. (Pre-challenge oracle queries) The adversary sends polynomially many queries. For each query x_i, the challenger checks if $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i. Else it samples $y_i \leftarrow \{0,1\}^\lambda$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$.
3. (Challenge) $\mathcal{A}$ sends m_0, m_1.
   Chall. chooses $r \leftarrow \{0,1\}^\lambda$. It checks if $(r, z) \in \mathcal{T}$. If not, it samples $z \leftarrow \{0,1\}^\lambda$, adds (r, z) to $\mathcal{T}$.
   Chall. sends $\mathsf{ct} = (\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_0)$ to $\mathcal{A}$.
4. (Post-challenge oracle queries) Handled same as pre-challenge oracle queries.
5. (Guess) $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_0$$

World-1:

1. (Setup) Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}(1^\lambda)$, sends pk to $\mathcal{A}$. It also maintains a table $\mathcal{T} = \varnothing$ for random oracle queries.
2. (Pre-challenge oracle queries) The adversary sends polynomially many queries. For each query x_i, the challenger checks if $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i. Else it samples $y_i \leftarrow \{0,1\}^\lambda$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$.
3. (Challenge) $\mathcal{A}$ sends m_0, m_1.
   Chall. chooses $r \leftarrow \{0,1\}^\lambda$. It checks if $(r, z) \in \mathcal{T}$. If not, it samples $z \leftarrow \{0,1\}^\lambda$, adds (r, z) to $\mathcal{T}$.
   Chall. sends $\mathsf{ct} = (\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_1)$ to $\mathcal{A}$.
4. (Post-challenge oracle queries) Handled same as pre-challenge oracle queries.
5. (Guess) $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_1$$

Next, we modify the above worlds to ensure that the challenge randomness r is chosen such that it is not equal to any of the pre-challenge random oracle queries. Since r is chosen uniformly at random, with overwhelming probability, it will be different from the random oracle queries in the pre-challenge phase (because there are only polynomially many random oracle queries). Looking ahead, this is needed because we want to use the TDF challenge in the challenge ciphertext.

Hyb-World-0:

1. (Setup) Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}(1^\lambda)$, sends pk to $\mathcal{A}$. It also maintains a table $\mathcal{T} = \varnothing$ for random oracle queries and set $\mathcal{S} = \varnothing$ for storing the random oracle queries.
2. (Pre-challenge oracle queries) The adversary sends polynomially many queries. For each query x_i, the challenger checks if $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i. Else it samples $y_i \leftarrow \{0,1\}^\lambda$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$ and x_i to $\mathcal{S}$.
3. (Challenge) $\mathcal{A}$ sends m_0, m_1.
   Chall. chooses $r \leftarrow \{0,1\}^\lambda \backslash \mathcal{S}$. It samples $z \leftarrow \{0,1\}^\lambda$, adds (r, z) to $\mathcal{T}$.
   Chall. sends $\mathsf{ct} = (\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_0)$ to $\mathcal{A}$.
4. (Post-challenge oracle queries) Handled same as pre-challenge oracle queries.
5. (Guess) $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},0}$$

Hyb-World-1:

1. (Setup) Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}(1^\lambda)$, sends pk to $\mathcal{A}$. It also maintains a table $\mathcal{T} = \varnothing$ for random oracle queries and set $\mathcal{S} = \varnothing$ for storing the random oracle queries.
2. (Pre-challenge oracle queries) The adversary sends polynomially many queries. For each query x_i, the challenger checks if $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i. Else it samples $y_i \leftarrow \{0,1\}^\lambda$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$ and x_i to $\mathcal{S}$.
3. (Challenge) $\mathcal{A}$ sends m_0, m_1.
   Chall. chooses $r \leftarrow \{0,1\}^\lambda \backslash \mathcal{S}$. It samples $z \leftarrow \{0,1\}^\lambda$, adds (r, z) to $\mathcal{T}$.
   Chall. sends $\mathsf{ct} = (\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_1)$ to $\mathcal{A}$.
4. (Post-challenge oracle queries) Handled same as pre-challenge oracle queries.
5. (Guess) $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p_{\mathsf{Hyb},1}$$

Observation 6.21. *$p_0 \approx p_{\mathsf{Hyb},0}$ and $p_1 \approx p_{\mathsf{Hyb},1}$.*

Proof. The number of pre-challenge queries is polynomial in λ, and r is drawn from an exponential-sized set. Therefore, with overwhelming probability, r is not equal to any of the pre-challenge random oracle queries. □

In the next two hybrids, the challenger chooses r and z during the challenge phase, but does not add (r, z) to $\mathcal{T}$. As a result, the adversary receives $(\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_0)$ or $(\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_1)$, and if the adversary queries the random oracle on r in the post-challenge query phase, then the challenger picks a fresh random string and sends it to the adversary.

Hyb-World-0′:

1. (Setup) Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}(1^\lambda)$, sends pk to $\mathcal{A}$. It also maintains a table $\mathcal{T} = \varnothing$ for random oracle queries and set $\mathcal{S} = \varnothing$ for storing the random oracle queries.
2. (Pre-challenge oracle queries) The adversary sends polynomially many queries. For each query x_i, the challenger checks if $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i. Else it samples $y_i \leftarrow \{0,1\}^\lambda$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$ and x_i to $\mathcal{S}$.
3. (Challenge) $\mathcal{A}$ sends m_0, m_1.
   Chall. chooses $r \leftarrow \{0,1\}^\lambda \backslash \mathcal{S}$. It samples $z \leftarrow \{0,1\}^\lambda$, does not add (r, z) to $\mathcal{T}$.
   Chall. sends $\mathsf{ct} = (\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_0)$ to $\mathcal{A}$.
4. (Post-challenge oracle queries) Handled same as pre-challenge oracle queries.
5. (Guess) $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p'_{\mathsf{Hyb},0}$$

Hyb-World-1′:

1. (Setup) Chall. chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}_{\mathsf{tdf}}(1^\lambda)$, sends pk to $\mathcal{A}$. It also maintains a table $\mathcal{T} = \varnothing$ for random oracle queries and set $\mathcal{S} = \varnothing$ for storing the random oracle queries.
2. (Pre-challenge oracle queries) The adversary sends polynomially many queries. For each query x_i, the challenger checks if $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i. Else it samples $y_i \leftarrow \{0,1\}^\lambda$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$ and x_i to $\mathcal{S}$.
3. (Challenge) $\mathcal{A}$ sends m_0, m_1.
   Chall. chooses $r \leftarrow \{0,1\}^\lambda \backslash \mathcal{S}$. It samples $z \leftarrow \{0,1\}^\lambda$, does not add (r, z) to $\mathcal{T}$.
   Chall. sends $\mathsf{ct} = (\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, r), z \oplus m_1)$ to $\mathcal{A}$.
4. (Post-challenge oracle queries) Handled same as pre-challenge oracle queries.
5. (Guess) $\mathcal{A}$ outputs b'.

$$\Pr[\mathcal{A} \text{ outputs } 0] = p'_{\mathsf{Hyb},1}$$

First, note that $p'_{\mathsf{Hyb},0} = p'_{\mathsf{Hyb},1}$ due to security of Shannon's one-time pad (the string z acts as a one-time pad since it is uniformly random, and not used anywhere else in the proof). However, we still need to argue that $p_{\mathsf{Hyb},b} \approx p'_{\mathsf{Hyb},b}$. For this, we will need to rely on the security of TDFs. Note that the only difference in the two hybrids is that the challenger records (r, z) in one case, and does not add (r, z) to $\mathcal{T}$ in the second case. The only way an adversary can distinguish between these two hybrid worlds is by querying the random oracle on r. However, if it sends such a query, then it must break TDF security.

Observation 6.22. *If there exists a p.p.t. adversary $\mathcal{A}$ such that $\left|p_{\mathsf{Hyb},b} - p'_{\mathsf{Hyb},b}\right|$ is non-negligible, then there exists a reduction algorithm $\mathcal{B}$ that breaks the security of the trapdoor function family.*

Proof. The reduction algorithm receives (pk, y^*) from the TDF challenger, it forwards pk to the adversary. For the pre-challenge random oracle queries, the reduction responds as in Hyb-World-b and Hyb-World-b′. When it receives challenge mesages m_0, m_1 from the adversary, it samples $z \leftarrow \{0,1\}^\lambda$, sends $(y^*, z \oplus m_b)$ to $\mathcal{A}$.

In the post-challenge phase, whenever the reduction receives a fresh query x_i, it checks if $\mathsf{Eval}_{\mathsf{tdf}}(\mathsf{pk}, x_i) = y^*$. If so, it sends x_i to the TDF challenger and wins the TDF game.

Analysis of reduction's winning probability: let r denote the pre-image of y^*.

$$\begin{aligned} p_{\mathsf{Hyb},b} &= \Pr\left[\begin{array}{c}\mathcal{A} \text{ outputs } 0 \text{ in Hyb-World-}b \\ \mathcal{A} \text{ queries on } r \text{ in post-challenge phase}\end{array}\right] \\ &\quad + \Pr\left[\begin{array}{c}\mathcal{A} \text{ outputs } 0 \text{ in Hyb-World-}b \\ \mathcal{A} \text{ does not query on } r \text{ in post-challenge phase}\end{array}\right] \\ p'_{\mathsf{Hyb},b} &= \Pr\left[\begin{array}{c}\mathcal{A} \text{ outputs } 0 \text{ in Hyb-World-}b' \\ \mathcal{A} \text{ queries on } r \text{ in post-challenge phase}\end{array}\right] \\ &\quad + \Pr\left[\begin{array}{c}\mathcal{A} \text{ outputs } 0 \text{ in Hyb-World-}b' \\ \mathcal{A} \text{ does not query on } r \text{ in post-challenge phase}\end{array}\right] \end{aligned}$$

Note that if the adversary does not query the random oracle on r in the post-challenge query phase, then Hyb-World-b and Hyb-World-b′ are identical.

$$\begin{aligned} &\Pr\left[\begin{array}{c}\mathcal{A} \text{ outputs } 0 \text{ in Hyb-World-}b \\ \mathcal{A} \text{ does not query on } r \text{ in post-challenge phase}\end{array}\right] \\ &= \Pr\left[\begin{array}{c}\mathcal{A} \text{ outputs } 0 \text{ in Hyb-World-}b' \\ \mathcal{A} \text{ does not query on } r \text{ in post-challenge phase}\end{array}\right] \end{aligned}$$

Therefore,

$$\begin{aligned} \left|p_{\mathsf{Hyb},b} - p'_{\mathsf{Hyb},b}\right| &\leq \Pr\left[\mathcal{A} \text{ queries on } r \text{ in post-challenge phase}\right] \\ &= \Pr\left[\text{Reduction wins TDF game}\right] \end{aligned}$$

and hence, if $\left|p_{\mathsf{Hyb},b} - p'_{\mathsf{Hyb},b}\right|$ is non-negligible, then the reduction algorithm wins the TDF game with non-negligible probability. □

Therefore, $p_0 \overset{\text{Obs. 6.21}}{\approx} p_{\mathsf{Hyb},0} \overset{\text{TDF}}{\approx} p'_{\mathsf{Hyb},0} \overset{\text{OTP}}{=} p'_{\mathsf{Hyb},1} \overset{\text{TDF}}{\approx} p_{\mathsf{Hyb},1} \overset{\text{Obs. 6.21}}{\approx} p_1$. □

(*) **Exercise 6.2.** *Recall, the security proof of Construction 6.11 was based on the* ***decisional*** *Diffie-Hellman assumption. Construct a heuristic PKE scheme whose security, in the random oracle model, can be based on the* ***computational*** *Diffie-Hellman assumption.*

Construction of TDFs based on a number-theoretic assumption

Trapdoor functions are a powerful extension of one-way functions. Note that the security requirement of TDFs is identical to the security requirement of OWFs : given evaluation on a random input, it is hard to find a preimage. However, the requirement of having a 'trapdoor' makes this primitive far more exotic. Rivest, Shamir and Adleman (RSA) gave the first TDF construction in 1978, based on the hardness of certain number-theoretic problem. Prior to their work, we didn't have any candidate constructions of TDFs. In fact, the RSA-TDF is actually a

trapdoor *permutation*. A trapdoor permutation is a trapdoor function where, for every public key pk, the evaluation function $\mathsf{Eval}(\mathsf{pk}, \cdot)$ is both injective and onto.

The starting point for RSA's TDF is an observation that we made towards the end of Section 6.2: given any element $g \in \mathbb{G}$, where $\mathbb{G}$ is a prime-order group of order q, and given any exponent $e \in \mathbb{Z}_q$, we can compute the e^{th} root of g efficiently. That is, we can find the unique group element y such that $y^e = g$. To do this, we will first find a number $d \in \mathbb{Z}_q$ such that $e \cdot d = 1 \bmod q$ (this can be computed efficiently using Extended Euclid's Algorithm), and then output $y = g^d$. Since $e \cdot d = k \cdot q + 1$, $y^e = g^{e \cdot d} = g^{k \cdot q + 1} = g$, because $g^q = 1_{\mathbb{G}}$. To summarise, we used two simple observations:

- given any $e \in \mathbb{Z}_q$, there exists a unique $d \in \mathbb{Z}_q$ such that $e \cdot d = k \cdot q + 1$ for some k, and this d can be computed efficiently.
- for any group element g, $g^q = 1_{\mathbb{G}}$ (the identity element in $\mathbb{G}$).

However, this computation would not have been possible if we did not know the order of the group! Therefore, a natural starting point is the following: can we have groups where the group operation is efficient, a random group element can be sampled efficiently, but finding the group order is *computationally hard*?

Consider a modulus N which is a product of two large distinct primes p and q. Let $\mathbb{Z}_N^*$ denote the set of all positive numbers less than N that are co-prime to N, that is,

$$\mathbb{Z}_N^* = \{\, x : 1 \leq x < N, \gcd(x, N) = 1 \,\}$$

Observation 6.23. $\left(\mathbb{Z}_N^*, \text{ mult. modulo } N\right)$ *is a commutative group.*

Proof. The identity element is 1, group closure, associativity and commutativity are immediate. The only thing we need to check is the existence of an inverse: for any $x \in \mathbb{Z}_N^*$, does there exist a $y \in \mathbb{Z}_N^*$ such that $x \cdot y = 1 \bmod N$? Using Extended Euclid's Algorithm, we can find a $y \in \mathbb{Z}_N$ such that $x \cdot y = 1 \bmod N$. Note that this y must be co-prime to N (since $x \cdot y + k \cdot N = 1$ for some integer k, there cannot be a number greater than 1 that divides both $x \cdot y$ and $k \cdot N$), hence $y \in \mathbb{Z}_N^*$. □

This group can be described by just giving the modulus N, the group operation is efficient, and group elements can be sampled efficiently (via rejection sampling). The order of this group is $N - p - q + 1$. How hard/easy is it to compute the order of this group, given N?

Observation 6.24. *Let $N = p \cdot q$ where p and q are primes. If there exists an efficient algorithm for computing $\left|\mathbb{Z}_N^*\right|$ given N, then there exists an efficient algorithm that, given N, can compute the prime factors of N.*

Proof. Let p, q be the prime factors of N. Given N and $\left|Z_N^*\right|$, we can compute $z = p + q$. Since $q = N/p$, we get a quadratic equation: $z = p + N/p$ (here z and N are known, p is the unknown), and this can be solved efficiently. □

Factoring large numbers is believed to be a hard computational problem, and using the reduction above, we get that computing $\left|\mathbb{Z}_N^*\right|$ is also computationally hard. The size of $\mathbb{Z}_N^*$ has a special name - *Euler's Totient Function* and is denoted by

$\phi(N)$. This brings us to the RSA conjecture. *RSA conjectured that for appropriately chosen primes p, q, given $N = p \cdot q$ and an exponent e* co-prime *to $\phi(N)$, computing the e^{th} root of a random number in $\mathbb{Z}_N^*$ is hard.* Again, since we are working with asymptotics, we assume there is an efficient algorithm RSA-Gen that takes as input the security parameter λ, and outputs two primes p, q.

We require e to be co-prime to $\phi(N)$ so that the e^{th} root is well-defined and unique. For e that is not co-prime to $\phi(N)$, a random group element may not have an e^{th} root.

Definition 6.25. *Let* RSA-Gen *be an efficient algorithm that takes as input a security parameter λ, and outputs two primes p, q. We say that* RSA-Gen *is RSA-hard if, for any p.p.t. adversary $\mathcal{A}$, the following probability is negligible:*

$$\Pr\left[\begin{array}{c}(p,q) \leftarrow \mathsf{RSA\text{-}Gen}\left(1^\lambda\right) \\ N = p \cdot q,\ y \leftarrow \mathbb{Z}_N^*,\ e \leftarrow \mathbb{Z}_{\phi(N)}^* \\ x \leftarrow \mathcal{A}(N,e,y) \ \textbf{\textit{and}}\ x^e \bmod N = y\end{array}\right]$$

◊

In practice, we use 512 or 1024 bit primes, and therefore the modulus N is 1024 or 2048 bits. This is nearly ten times more expensive (in terms of ciphertext length) than symmetric key encryption, where we used AES with 128 or 256 bit keys.

Given the RSA assumption, the TDF construction is somewhat immediate.

Construction 6.26 (Trapdoor Permutation from RSA)**.**

- $\mathsf{Setup}\left(1^\lambda\right)$: *Let $(p,q) \leftarrow \mathsf{RSA\text{-}Gen}\left(1^\lambda\right)$ and $N = p \cdot q$. The setup algorithm samples $e \leftarrow \mathbb{Z}_{\phi(N)}^*$, sets $\mathsf{pk} = (N,e)$. The secret key $\mathsf{sk} = (N,d)$, where d is the unique integer in $\mathbb{Z}_{\phi(N)}^*$ such that $e \cdot d = 1 \bmod \phi(N)$. The domain and co-domain of the function is $\mathbb{Z}_N^*$.*
- $\mathsf{Eval}(\mathsf{pk} = (N,e), x \in \mathbb{Z}_N^*)$: *The output of the evaluation is $x^e \bmod N$.*
- $\mathsf{Inv}(\mathsf{sk} = d, y)$: *The inversion algorithm outputs $y^d \bmod N$.*

Correctness: Let $\mathsf{pk} = (N,e)$, $\mathsf{sk} = d$, $x \in \mathbb{Z}_N^*$, $y = x^e \bmod N$. The inversion algorithm outputs $z = y^d \bmod N$, and we need to prove that $z = x$.

$$\begin{aligned} z &= (x^e \bmod N)^d \bmod N \\ &= x^{e \cdot d} \bmod N \\ &= x^{1 + k\phi(N)} \bmod N \\ &= \left((x \bmod N) \cdot \left(x^{k\phi(N)} \bmod N\right)\right) \bmod N \end{aligned}$$

Since $x \in \mathbb{Z}_N^*$, $x^{\phi(N)} \bmod N = 1$, and therefore $z = x$.

Note: even if $x \in \mathbb{Z}_N$, one can show that $(x^e \bmod N)^d \bmod N = x$. However, the above proof does not work since we cannot conclude that $x^{\phi(N)} \bmod N = 1$ (why?).

The one-wayness of the TDF follows from the RSA assumption. ◊

Putting everything together, the following is a public-key encryption scheme whose security can be proven in the random oracle model, assuming the RSA problem is hard. Combining Construction 6.26 and Construction 6.18, we get a PKE scheme for λ bit messages. Combining this with Construction 6.4, we get a PKE scheme for long messages.

Construction 6.27 (RSA-based PKE)**.** *Let $H : \mathbb{Z}_N \to \{0,1\}^\lambda$ be a hash function (that will be modeled as a random oracle in the proof), let $\mathcal{E}_{\text{ske}} = (\mathsf{Enc}_{\text{ske}}, \mathsf{Dec}_{\text{ske}})$ be a secret-key encryption scheme with key space $\{0,1\}^\lambda$ and message space $\{0,1\}^*$.*

- $\mathsf{Setup}\left(1^\lambda\right)$: *Sample $(p,q) \leftarrow \mathsf{RSA\text{-}Gen}\left(1^\lambda\right)$. Let $N = p \cdot q$. Sample $e \leftarrow \mathbb{Z}^*_{\phi(N)}$, and let d be the unique integer in $\mathbb{Z}^*_{\phi(N)}$ such that $e \cdot d = 1 \bmod \phi(N)$. Set $\mathsf{pk} = (N,e)$ and $\mathsf{sk} = (N,d)$.*
- $\mathsf{Enc}(\mathsf{pk}, m)$: *Choose $r \leftarrow \mathbb{Z}_N$, $k \leftarrow \{0,1\}^\lambda$. Let $\mathsf{ct}_1 = r^e \bmod N$, $\mathsf{ct}_2 = H(r) \oplus k$, $\mathsf{ct}_3 \leftarrow \mathsf{Enc}_{\text{ske}}(k, m)$. The final ciphertext is $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$.*
- $\mathsf{Dec}(\mathsf{sk} = (N,d), \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3))$: *Let $r = \mathsf{ct}_1^d \bmod N$, $k = \mathsf{ct}_2 \oplus H(r)$, output $\mathsf{Dec}_{\text{ske}}(k, \mathsf{ct}_3)$.*

The above scheme can be optimized further. Instead of choosing k and setting $\mathsf{ct}_2 = H(r) \oplus k$, one can use $H(r)$ as the SKE key. Therefore, in this optimized scheme, the PKE encryption of m is $(r^e \bmod N, \mathsf{Enc}_{\text{ske}}(H(r), m))$

(*) Exercise 6.3. *Prove security of the optimized construction in the random oracle model, assuming $\mathcal{E}_{\text{ske}}$ is CPA secure SKE scheme and RSA assumption holds.*

$\Diamond$

6.5 *Incorrect Instantiations of PKE*

Lecture 23: October 31st, 2023

Here are some incorrect instantiations of PKE systems based on RSA:

- The most naive mistake is to use RSA without any randomization. More formally, to encrypt a message m using public key (N,e), one can deterministically encode m as a number $z \in \mathbb{Z}_N$, and then output $z^e \bmod N$. This method cannot be CPA secure as it is deterministic. This method of encryption is referred to as *textbook RSA*.
- In RSA, it is very important to choose the primes carefully. Here are some incorrect ways of doing it:
  - Choosing the first prime p at random, and then outputting the prime q closest to p. The modulus N is set to be $p \cdot q$. However, this is broken since an adversary can compute the factors of N by outputting the primes closest to $\sqrt{N}$.
  - If the primes are not chosen from a distribution with sufficient entropy, then it is easy to find the prime factors of N.

- Another common source of error is choosing an exponent e such that the inverse d is a small number. While this would ensure faster decryption, the RSA assumption no longer holds. Similarly, choosing a small public exponent e also leads to vulnerabilities in certain systems.

6.6 *PKE History and Notes*

The following are some selected landmark moments in the history of public key cryptography. This section is not part of the syllabus, but some of the lectures/references linked below are very insightful.

- 1969: Clifford Cocks, a mathematician at the British intelligence agency (Government Communications Headquarters - GCHQ) gave the first public key encryption scheme. However, due to a confidentiality agreement with GCHQ, this invention remained hidden until the late 90s — this contribution remained classified.

- 1976: Whitfield Diffie and Martin Hellman proposed their key exchange protocol, based on a group-theoretic problem which we now call the (Decisional) Diffie-Hellman assumption. Around the same time, Ralph Merkle also proposed a key exchange protocol (although that only guaranteed security against quadratic time adversaries). Diffie and Hellman also proposed the notion of public key encryption, digital signatures and trapdoor functions. Their landmark paper [DH76]opened up new directions in cryptography, and they received the Turing Award in 2015. Check out their Turing Award interviews: Hellman, Diffie. Diffie and Hellman's ideas were so powerful that they attracted the attention of NSA and the American government, who compared their ideas to *modern weapons technologies*. You can read more about the first *crypto wars* here.

- 1978: Ron Rivest, Adi Shamir and Leonard Adleman [RSA83] gave the first public key encryption scheme (and the first digital signature scheme), based on computational number-theoretic problem (which is now referred to as the RSA assumption). Strictly speaking, they constructed a trapdoor permutation (which was believed to be very close to public key encryption). Their schemes are the most widely used public-key primitives, and they received the Turing Award for their contributions in 2002. Check out their Turing Award lectures: Adleman, Rivest, Shamir.

- 1982: Shafi Goldwasser and Silvio Micali proposed the notion of probabilistic encryption [GM82], and gave a construction based on a different number-theoretic problem involving composite numbers. They made numerous other influential contributions to theoretical cryptography, and were awarded the Turing Award in 2012. Their Turing Award interviews are available here: Goldwasser, Micali.

  In the same year (1982), Andrew Yao showed [Yao82] how to use a TDF to build CPA-secure PKE (without random oracles). Yao also received the Turing Award (in 2000) for several contributions to theory of computation, complexity theory and cryptography.

Chapter Summary and References

- We introduced the notion of public key encryption, and saw the definition for security against passive attacks (see Definition 6.3). We concluded that it suffices to consider only one challenge query (instead of polynomially many challenge queries as in SKE); see Lemma 6.1. In practice, when we want to encrypt a mesage using a public key, instead of directly encrypting the message (which can be very expensive), we sample an SKE key, encrypt the SKE key using the public key, and then encrypt the message using the SKE key; see Claim 6.5.

- Next, we saw a key exchange protocol given by Whitfield Diffie and Martin Hellman (Section 6.2). The security of this key exchange protocol has been well studied and cryptanalyzed, and it is now a popular cryptographic assumption (called Decisional Diffie-Hellman - DDH). We saw a public key encryption scheme whose security can be based on this assumption (Construction 6.11, Claim 6.12). The ciphertexts derived from this scheme also hide the public key used to encrypt the message, this property is called anonymity (see Definition 6.13).

  This encryption scheme has nice homomorphic properties, which were used for building a private information retrieval system (Construction 6.14.

- The most widely used public key encryption scheme was proposed by Rivest, Shamir and Adleman. They showed how to use a number-theoretic problems over composite numbers (now referred to as the RSA assumption - Definition 6.25 - another widely cryptanalyzed assumption) to build a cryptographic primitive called trapdoor functions (TDFs - Definition 6.15). Trapdoor functions, while syntactically similar to PKE, do not immediately give us a PKE scheme. We saw a heuristic construction of PKE from TDFs, which can be proven secure in the random oracle model (Construction 6.18, Claim 6.20).

Textbook References: Boneh-Shoup (v6, [BS23]): Sections 11.1, 11.2, 11.3, 10.4, 10.5, 11.5, 10.2, 10.3, 11.4.

Suggested textbook exercises: 10.14, 10.27, 11.8, 11.10, 11.12, 11.13

7 PKE Secure Against Active Attacks

In the last section, we discussed security against passive attacks, and saw two popular encryption schemes - the Elgamal scheme and the RSA scheme. In the early 90s, various cryptographic schemes and protocols were standardized. One of these standards, PKCS v1.5, was widely adopted in practice. Below, we discuss the PKCS v1.5 encryption standard, proposed for encrypting $t < 117$ byte messages. Typically, t was either 16, 32 or 48 (since we only need to RSA-encrypt the AES key; the AES key can then be used to encrypt the actual message via hybrid encryption).

- The RSA modulus N is a 1024-bit number. Given the t-byte message m, choose $\ell = 128 - t - 3$ random non-zero bytes $r_1, r_2, \ldots, r_\ell$. Consider the 128-byte string x defined below:

$$x = \underbrace{\texttt{0x00}}_{1\text{ byte}}\ \underbrace{\texttt{0x02}}_{1\text{ byte}}\ \underbrace{r_1\, r_2\, \ldots\, r_\ell}_{128-t-3\text{ bytes}}\ \underbrace{\texttt{0x00}}_{1\text{ byte}}\ \underbrace{m}_{t\text{ bytes}}$$

  That is, the first byte is 0x00, the next byte is 0x02, the next $128 - t - 3$ bytes are non-zero bytes, the next byte is 0x00, and finally we have the t byte message.

- The string x, when expressed as a number, will be less than N (since the leading byte is 0x00). Set $x^e \bmod N$ as the encryption of m.

To decrypt, first compute $\text{ct}^{\text{sk}} \bmod N$. This number, when expressed as a string of 128 bytes, must have the first two bytes as 0x00 and 0x02 respectively (otherwise output `'padding error'`). If it is not a padding error, find the first 0x00 byte after the 0x02 byte, and output the remaining bytes as the decryption.

The CPA security of this scheme does not follow immediately from our discussion in the previous section. We showed that RSA is a trapdoor permutation which, as per Definition 6.15, ensures that the adversary cannot recover the entire preimage. However, does the RSA assumption also ensure that the adversary cannot learn any of the lower order bits? It turns out that it does (and therefore the above construction is CPA secure).

This encryption scheme was used for several communication protocols, including the SSL protocol. However, in 1998, David Bleichenbacher showed a devastating attack against the SSL protocol, and the fundamental vulnerability was that the PKCS v1.5 encryption standard is CPA secure, but it does not prevent *active attacks*. Bleichenbacher's attack works as follows:

- Suppose a server and a client are interacting. The client encrypts a 48 byte string m using the server's public key. The server, on receiving the ciphertext, decrypts the ciphertext. If the recovered string has a padding error, the server sends `'padding error'`. Else, it continues the conversation with the client by sending a non-error message.

- The adversary intercepts a ciphertext sent by the client. It then repeatedly tampers the ciphertext, and sends these tampered ciphertexts to the server. The server either sends a padding error, or sends a non-error message.

Bleichenbacher's PKCS attack will be discussed in detail in the next assignment.

Similar to the CBC padding oracle attack discussed in Section 5, the adversary only needs one bit of information — whether it is valid padding or not — and this is sufficient to recover the entire message m!

The moral of this story is that security against passive attacks (CPA security) is not sufficient when cryptosystems are deployed in the wild. We need to handle active attacks.

7.1 *Defining Active Attacks for PKE*

Recall, in symmetric key encryption, we captured active attacks using the notion of authenticated encryption. A symmetric key encryption scheme is an authenticated encryption scheme if it is both CPA secure, and also satisfies ciphertext integrity. Equivalently, it should satisfy both CCA security and plaintext integrity.

However, in the public key setting, an encryption cannot satisfy plaintext/ciphertext integrity, since the adversary has the public key, and can always produce an encryption of a message of its choice. Therefore, we aim for security against *chosen ciphertext attacks* (CCA). As we will see, CCA security captures padding oracle attacks, malleability attacks, and therefore any public key cryptosystem deployed in practice must be at least CCA secure.

Chosen Ciphertext Attacks (CCA) for PKE

- (Setup) Challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}\left(1^\lambda\right)$ and bit $b \leftarrow \{0,1\}$.
- (Pre-challenge queries) Adversary is allowed polynomially many decryption queries. On sending a ciphertext ct_j, it receives $y_j \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_j)$.
- (Challenge) Adversary sends two messages m_0, m_1. The challenger computes $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ and sends ct^*.
- (Post-challenge queries) Adversary is allowed polynomially many post-challenge decryption queries. On sending a ciphertext $\mathsf{ct}'_j \neq \mathsf{ct}^*$, it receives $y'_j \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}'_j)$.
- The adversary finally outputs its guess b', and wins the game if $b = b'$.

Figure 25: The security game for capturing chosen ciphertext attacks against public key encryption schemes. This is defined with respect to an encryption scheme $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$. The game is similar to the CPA security game, except that the adversary also gets to make decryption queries.

Note that the post-challenge decryption query should be different from the challenge ciphertext (otherwise the adversary can trivially win the security game). The CCA security game captures padding oracle attacks and malleability attacks.

(*) Exercise 7.1. *Show that the Elgamal encryption scheme (Construction 6.11) and the RSA encryption scheme (Construction 6.27) do not satisfy CCA security. Both these schemes are susceptible to malleability attacks.*

7.2 *Constructing CCA secure PKE*

In symmetric key encryption, we saw that Encrypt-then-MAC is one way to ensure CCA security. Given a CPA scheme and a MAC scheme, we can first encrypt the message using the CPA scheme's encryption key, and then sign the resulting ciphertext using the MAC key. However, the same approach cannot be used for public key encryption. The MAC key needs to be used by both the encryptor (for signing the ciphertext) and the decryptor (for verifying the signature), but it is not clear how to achieve this. The following interesting approach was proposed in class: the encryptor chooses a MAC key k, then encrypts $m \| k$ using the CPA secure PKE scheme, computes a signature on this ciphertext using k, and sends the ciphertext and signature. While this construction is not immediately broken, we cannot prove security assuming just CPA security of the PKE scheme and UFCMA security of MAC. This is because the signature is computed using key k on a string that depends on k.

There are also other issues, even if we ignore the key dependent message. Please discuss with me if you're interested.

Can we modify the RSA construction (Construction 6.27) to prevent malleability attacks? The following candidates were proposed in class.

Construction 7.1 (RSA-based CCA Construction: Attempt 1)**.** *Let $H : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a hash function, consider the following encryption scheme with message space $\{0,1\}^\lambda$. Let $\mathsf{bin}(x)$ denote the λ-bit binary representation of x.*

- $\mathsf{Setup}\left(1^\lambda\right)$: *Sample* $(p,q) \leftarrow \mathsf{RSA\text{-}Gen}\left(1^\lambda\right)$. *Let* $N = p \cdot q$. *Sample* $e \leftarrow \mathbb{Z}^*_{\phi(N)}$, *and let* d *be the unique integer in* $\mathbb{Z}^*_{\phi(N)}$ *such that* $e \cdot d = 1 \bmod \phi(N)$. *Set* $\mathsf{pk} = (N,e)$ *and* $\mathsf{sk} = (N,d)$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: *Choose* $r \leftarrow \mathbb{Z}_N$. *Let* $\mathsf{ct}_1 = r^e \bmod N$, $\mathsf{ct}_2 = H(\mathsf{bin}(r)) \oplus m$, $\mathsf{ct}_3 = H(\mathsf{bin}(r) \oplus m)$. *The final ciphertext is* $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$.
- $\mathsf{Dec}(\mathsf{sk} = (N,d), \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3))$: *Let* $r = \mathsf{ct}_1^d \bmod N$, $m = \mathsf{ct}_2 \oplus H(\mathsf{bin}(r))$. *If* $H(\mathsf{bin}(r) \oplus m) = \mathsf{ct}_3$, *output* m.

This proposal uses the third ciphertext component to enforce a consistency check on the ciphertext. Unfortunately, it is not CCA secure. The adversary can sends $0^n, 1^n$ as the challenge messages. If 0^n is encrypted, then $\mathsf{ct}_2 = \mathsf{ct}_3$, else $\mathsf{ct}_2 \neq \mathsf{ct}_3$. ◊

Construction 7.2 (RSA-based CCA Construction: Attempt 2)**.** *Let $H : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a hash function, consider the following encryption scheme with message space $\{0,1\}^\lambda$. Let $\mathsf{bin}(x)$ denote the λ-bit binary representation of x.*

- $\mathsf{Setup}\left(1^\lambda\right)$: *Sample* $(p,q) \leftarrow \mathsf{RSA\text{-}Gen}\left(1^\lambda\right)$. *Let* $N = p \cdot q$. *Sample* $e \leftarrow \mathbb{Z}^*_{\phi(N)}$, *and let* d *be the unique integer in* $\mathbb{Z}^*_{\phi(N)}$ *such that* $e \cdot d = 1 \bmod \phi(N)$. *Set* $\mathsf{pk} = (N,e)$ *and* $\mathsf{sk} = (N,d)$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: *Choose* $r \leftarrow \mathbb{Z}_N$. *Let* $\mathsf{ct}_1 = r^e \bmod N$, $\mathsf{ct}_2 = H(\mathsf{bin}(r) \oplus m)$.

The final ciphertext is $(\mathsf{ct}_1, \mathsf{ct}_2)$.

While this construction takes care of the malleability issue, we cannot recover the message from the ciphertext, since H is not invertible. ◊

Lecture 24: November 3rd, 2023

We will now discuss how to fix Construction 6.27. Note that Construction 6.27 is using $H(r)$ as the key for one-time padding the message. Since the one-time pad is malleable, so is the resulting construction. To prevent this malleability attack, we should encrypt the message using an authenticated encryption scheme, with $H(r)$ as the key.

Construction 7.3 (RSA-based CCA secure PKE)**.** *Let* $\mathcal{E}_{\text{ske}} = (\mathsf{Enc}_{\text{ske}}, \mathsf{Dec}_{\text{ske}})$ *be a symmetric key authenticated encryption scheme with message space* $\{0,1\}^*$ *and key space* $\{0,1\}^\lambda$*. Let* $H : \mathbb{Z}_N \to \{0,1\}^\lambda$ *be a deterministic hash function (which will be modeled as a random oracle in the proof).*

- $\mathsf{Enc}(\mathsf{pk} = (N, e), m)$: *Choose* $r \leftarrow Z_N$*, compute* $\mathsf{ct}_1 = r^e \bmod N$*,* $k = H(r)$*,* $\mathsf{ct}_2 = \mathsf{Enc}_{\text{ske}}(k, m)$.
- $\mathsf{Dec}(\mathsf{sk} = (N, d), (\mathsf{ct}_1, \mathsf{ct}_2))$: *Compute* $r = \mathsf{ct}_1^d \bmod N$*,* $k = H(r)$*. Output* $\mathsf{Dec}_{\text{ske}}(k, \mathsf{ct}_2)$.

◊

Correctness is immediate, let us consider the CCA security proof.

Lemma 7.4. *Assuming $\mathcal{E}_{\text{ske}}$ is a CCA secure encryption scheme, and assuming the RSA assumption holds, Construction 7.3 is a public key encryption scheme that is CCA secure in the random oracle model.*

Proof idea: The main challenge in this proof is responding to decryption queries. Recall, in the case of symmetric key encryption, handling decryption queries was easy: if the ciphertext was previously sent by the challenger in response to an encryption query, then we know the decryption of the message; else we send $\perp$. However, since this is a public key encryption scheme, the adversary can generate an encryption of any message of its choice, and send the encryption as a decryption query. In the CCA experiment, the challenger can compute the decryption because it has the RSA secret key d. However, looking ahead, the reduction breaking RSA security will not have the RSA secret key. How does it respond to decryption queries?

The key idea here is that the reduction responds to decryption queries issued by the adversary. As a result, if the adversary queries for a decryption of $\mathsf{ct} = (r^e, \mathsf{Enc}_{\text{ske}}(H(r), m))$, then it should have queried the random oracle on the input r. The reduction maintains a table $\mathcal{T}$, where it has all random oracle queries, together with the corresponding responses. The reduction checks if there is an entry (r, z) in the table such that $r^e = \mathsf{ct}_1$. If so, it uses z to decrypt ct_2.

There is one minor subtlety: what if the adversary first queries for the decryption of ct, and then queries the random oracle on r? To handle this, the reduction should sample a random z, use z to decrypt ct_2, and store (ct_1, z) in a separate

we will see in the formal proof that the reduction needs to store only one table.

table $\mathcal{S}$. Later, when it receives a random oracle query for r, it checks if $r = \mathsf{ct}_1$ for some entry (ct_1, z) in $\mathcal{S}$. If so, it adds (r, z) to $\mathcal{T}$ and sends z as the random oracle response.

Proof. The proof proceeds via a sequence of hybrid games.

Game 0: This is the original semantic security game in the random oracle model.

- (Setup phase) The challenger chooses $\mathsf{pk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Pre-challenge queries)
  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}^{1/e}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}^{1/e}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.
- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$. Next, it checks if $(x^*, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.
  If no $(x^*, y) \in \mathcal{T}$, it chooses $y \leftarrow \{0,1\}^n$, adds (x^*, y) to $\mathcal{T}$, sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.
  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.
- (Post-challenge queries)
  - (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries
  - (Post-challenge Decryption queries) Same as pre-challenge decryption queries, except that the challenger outputs $\perp$ if the decryption query is equal to the challenge ciphertext.

Game 1: This is similar to the previous game, except that the challenger stores (x^e, y) in $\mathcal{T}$ instead of (x, y). Note that since everything else is identical, the game is identical in the adversary's view. However, this change is important because instead of computing the e^{th} root (which requires the secret key component $d = 1/e$), the challenger only computes the e^{th} power (which is an easy operation).

- (Setup phase) The challenger chooses $\mathsf{pk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Pre-challenge queries)

  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i^e, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i^e, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.

- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$. Next, it checks if $((x^*)^e, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.

  If no $((x^*)^e, y) \in \mathcal{T}$, it chooses $y \leftarrow \{0,1\}^n$, adds $((x^*)^e, y)$ to $\mathcal{T}$, sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.

  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

- (Post-challenge queries)

  - (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries
  - (Post-challenge Decryption queries) Same as pre-challenge decryption queries, except that the challenger outputs $\perp$ if the decryption query is equal to the challenge ciphertext.

Game 2: In this game, the challenger does not use the table for the challenge ciphertext. It just samples a random key k and uses it for the challenge ciphertext, but does not add $((x^*)^e, k)$ to the table $\mathcal{T}$.

- (Setup phase) The challenger chooses $\mathsf{pk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.

- (Pre-challenge queries)

  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i^e, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i^e, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.

- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$.

  ~~Next, it checks if $((x^*)^e, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.~~

  It chooses $k \leftarrow \{0,1\}^n$, ~~adds $((x^*)^e, y)$ to $\mathcal{T}$,~~ sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(k, m_b^*)$.

  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

- (Post-challenge queries)

– (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries. Note: if there is a random oracle query for x^*, then the adversary samples a fresh string y^*, adds (x^*, y^*) to $\mathcal{T}$ and sends y^*. This is because $((x^*)^e, k)$ was not added to $\mathcal{T}$ during the challenge phase.

– (Post-challenge Decryption queries) Let ct_i be the decryption query. If $\mathsf{ct}_i = \mathsf{ct}^*$, then output $\perp$.
If $\mathsf{ct}_{i,1} = \mathsf{ct}_1^*$ but $\mathsf{ct}_{i,2} \neq \mathsf{ct}_2^*$, the challenger sends $\mathsf{Dec}_{\mathsf{ske}}(\mathsf{ct}_{i,2}, k)$.[3]
If $\mathsf{ct}_{i,1} \neq \mathsf{ct}_1^*$, it proceeds similar to the pre-challenge decryption queries. It first checks if $(\mathsf{ct}_{i,1}, y)$ is present in $\mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$ and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.

Analysis: First, observe that Game 0 and Game 1 are equivalent from the adversary's perspective. In one case, the challenger adds (x, y) to the table, in the other case it adds (x^e, y) to the table. Therefore, if an adversary wins with non-negligible advantage in Game 0, then it wins with non-negligible advantage in Game 1.

Next, let us consider the differences between Game 1 and Game 2.

- The first difference is in the challenge phase. In Game 1, the challenger first samples $x^* \leftarrow \mathbb{Z}_N$, then checks if $((x^*)^e, y) \in T$. In Game 2, the challenger simply picks a random k and uses it for computing ct_2^*.
- The second difference is in the post-challenge queries. Note, in Game 1, after the challenge phase, there exists a tuple $((x^*)^e, y) \in \mathcal{T}$. If there is a post-challenge random oracle query on x^*, the challenger sends y. If there is a post-challenge decryption query for $(\mathsf{ct}_1^*, \mathsf{ct}_{i,2})$, then the challenger uses y to decrypt $\mathsf{ct}_{i,2}$. In Game 2, the challenger sends $((x^*)^e, \mathsf{Enc}_{\mathsf{ske}}(k, m_b))$, 'stores' k separately, but does not add $((x^*)^e, k)$ to $\mathcal{T}$. As a result, if there is a post-challenge random oracle query for x^*, the challenger samples a uniformly random string y^* and sends this to the adversary. However, if there is a decryption query for $(\mathsf{ct}_1^*, \mathsf{ct}_{i,2})$, the challenger uses k to decrypt $\mathsf{ct}_{i,2}$. Therefore, the difference is in the post-challenge random oracle response to x^*.

Intuitively, the first event will happen with very low probability (since x^* is chosen at random during the challenge phase). The second one will also happen with low probability, and this argument relies on the hardness of RSA problem. If an adversary has significant difference in winning probabilities in the two games, then it must break the RSA assumption.

Formal proof: Let p_b denote the probability of adversary $\mathcal{A}$ winning in Game b.

Claim 7.5. *Suppose there exists a p.p.t. adversary $\mathcal{A}$ such that $p_1 - p_2 = \epsilon$. Then there exists a p.p.t. algorithm $\mathcal{B}$ that solves the RSA problem with probability ϵ.*

[3] This is needed because we have not added (ct_1^*, k) to $\mathcal{T}$.

Proof. The reduction algorithm receives (N, e, y) from the RSA challenger. It sends $\mathsf{pk} = (N, e)$ to the adversary. The reduction also maintains a table $\mathcal{T}$, which is initially empty.

Next, the adversary makes pre-challenge random oracle, and pre-challenge decryption queries. For each RO query x_i, the reduction algorithm does the following:

- If there exists an entry (x_i^e, y_i) in $\mathcal{T}$, then the reduction algorithm sends y_i.
- If there exists no entry corresponding to x_i in $\mathcal{T}$, the reduction algorithm samples $y_i \leftarrow \{0,1\}^n$, adds (x_i^e, y_i) to $\mathcal{T}$ and sends y_i to $\mathcal{A}$. It also checks if $x_i^e = y$. If so, it sends x_i to the RSA challenger (and wins the RSA game).

For the challenge phase, the reduction algorithm receives challenge messages m_0^*, m_1^* from the adversary. It picks a uniformly random string $k \leftarrow \{0,1\}^n$, sets $\mathsf{ct}_1^* = y$. Next, it picks a random bit $b \leftarrow \{0,1\}$, computes $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(k, m_b^*)$ and sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

Note: If the e^{th} root of y was queried in one of the random oracle queries, then the reduction has already won the RSA game. If it was not queried, then Game 1 and Game 2 are identical up to the challenge phase (and the reduction perfectly simulates the two games up to this point).

The adversary then makes post-challenge random oracle queries. For each RO query x_i, the reduction algorithm checks if $x_i^e = y$. If so, it sends x_i to the RSA challenger. Else, it checks if there exists an entry $(x_i^e, y_i) \in \mathcal{T}$. If so, it sends y_i to $\mathcal{A}$. If there is no such entry, then it samples $y_i \leftarrow \{0,1\}^n$, sends y_i to $\mathcal{A}$ and adds (x_i^e, y_i) to $\mathcal{T}$.

Similar to what is mentioned above, if the adversary does not make a random oracle query on input $y^{(1/e)}$, then Game 1 and Game 2 are identical. Therefore, if the winning probabilities are different in the two games, then it must query the random oracle on input $y^{(1/e)}$, in which case the reduction wins the RSA game. □

Claim 7.6. *Suppose there exists a p.p.t. adversary $\mathcal{A}$ that wins in Game 2 with probability ϵ. Then there exists a reduction algorithm $\mathcal{B}$ that wins the CCA security game against $\mathcal{E}_{\mathsf{ske}}$ with probability ϵ.*

Proof. The proof follows from the simple observation that in Game 2, string k is only used in the challenge phase (for computing the challenge ciphertext) and in the post-challenge decryption query phase (for answering decryption queries where the first component is identical to ct_1^*).

The reduction algorithm chooses (N, e) and sends it to the adversary. The reduction maintains a table $\mathcal{T}$ which is initially empty. On receiving a random oracle query x_i, the reduction checks if there exists (x_i^e, y_i) in the table. If so, it sends y_i. Else, it chooses $y_i \leftarrow \{0,1\}^n$, adds (x_i^e, y_i) to $\mathcal{T}$ and sends y_i to $\mathcal{A}$.

When the adversary sends (m_0, m_1) as the challenge messages, the reduction sends (m_0, m_1) to the $\mathcal{E}_{\mathsf{ske}}$ challenger. It receives ct from the challenger. The reduction algorithm chooses $x \leftarrow \mathbb{Z}_N$, sets $\mathsf{ct}_1^* = x^e \bmod N$, $\mathsf{ct}_2^* = \mathsf{ct}$ and sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

The post-challenge random-oracle queries are handled similar to the pre-challenge ones. For the post-challenge decryption query $(\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$, if $\mathsf{ct}_{i,1} = \mathsf{ct}_1^*$,

the reduction sends $\mathsf{ct}_{i,2}$ to the $\mathcal{E}_{\mathsf{ske}}$ challenger, and forwards the decryption to the adversary.

Note: In Game 2's challenge phase, the challenger chooses $x \leftarrow \mathbb{Z}_N$, $k \leftarrow \{0,1\}^n$, but does not add (x,k) to $\mathcal{T}$. As a result, if the adversary queries the random oracle on this x, it receives a fresh random string y. This is important for our reduction here.

Finally, the adversary sends its guess b', which the reduction forwards to the challenger. The winning probability of $\mathcal{A}$ in Game 2 is equal to the winning probability of the reduction against $\mathcal{E}_{\mathsf{ske}}$ (why?). □

□

7.3 *PKE-CCA History and Notes*

- The definition of CCA security for public key encryption was given by Moni Naor and Moti Yung [NY90] and Charles Rackoff and Daniel Simon [RS91]. They gave a weaker security notion (called CCA-1, where only pre-challenge decryption queries are allowed), and also gave a construction *in the standard model*. The Naor-Yung construction is very elegant, using *noninteractive zero-knowledge proofs*. Although it is very inefficient practically, the proof technique has found several other applications. The full CCA definition (also referred to as CCA-2), as well as a construction in the standard model, was given by Danny Dolev, Cynthia Dwork and Moni Naor [DDN03]. Note that the first CCA definitions came much before Bleichenbacher's attack.

- Daniel Bleichenbacher demonstrated the practical CCA attack in 1998 [Ble98]. This attack has immense impact on cryptography and security. Researchers have shown Bleichenbacher-style attacks on several other systems. This attack is also the reason why CCA security is the 'gold standard' for encryption schemes. For instance, see the recent NIST call for post-quantum cryptosystems - they want CCA security in the random oracle model:

  > *NIST intends to standardize one or more schemes that enable "semantically secure" encryption or key encapsulation with respect to adaptive chosen ciphertext attack, for general use. This property is generally denoted IND-CCA2 security in academic literature.*

- The first practical CCA secure encryption scheme in the standard model was given by Ronald Cramer and Victor Shoup [CS98]. See also Victor's excellent writeup on why chosen ciphertext security matters.

- In practice, we use schemes that are proven CCA secure in the random oracle model. After Bleichenbacher's attack, the PKCS standards were updated. Today, a widely used candidate is *RSA with optimal asymmetric encryption padding* (RSA-OAEP). This was proposed by Mihir Bellare and Philip Rogaway [BR94]. Similar to the PKCS v1.5 scheme, here the ciphertext consists of only one component. However, it is provably secure in the random oracle model.

- A major open question in theoretical public key cryptography is whether CPA security implies CCA security in the standard model. Quite often, constructing a CPA secure scheme is much easier than constructing a CCA secure scheme. Therefore, it would be nice if we could take any CPA secure scheme, and convert that into a CCA secure scheme. However, currently we don't know how to do this, and it is also not clear if there are any barriers towards such a transformation.
- In the random oracle model, we have an efficient transformation from CPA security to CCA security. This was given by Eiichiro Fujisaki and Tatsuaki Okamoto [FO99]. The leading proposals for the post-quantum encryption schemes first build a post-quantum CPA secure encryption scheme, and then use the Fujisaki-Okamoto transformation to obtain a post-quantum CCA secure encryption scheme in the random oracle model.

Chapter Summary and References

- We saw that CPA security may not suffice for real-world applications. One of the cryptographic standards in the 90s, PKCS v1.5, was CPA secure, but there was an active attack against this encryption scheme.
- Next, we defined security that captures active attacks — security against chosen ciphertext attacks (CCA). The constructions discussed in the previous section were not CCA secure.
- Finally, we saw a heuristic construction that can be proven CCA secure in the random oracle model, assuming hardness of RSA.

Textbook References: Boneh-Shoup (v6, [BS23]): 12.1, 12.2, 12.3, 12.8.3.

Suggested textbook exercises: 12.2, 12.4, 12.5, 12.16.

8 Digital Signatures

Lecture 25: November 7th, 2023

In the last few lectures, we studied public key encryption, and saw various constructions for the same. We discussed the 'gold-standard' security notion for public key encryption (CCA security) and even saw a construction that's secure in the random oracle model. However, this still does not solve one fundamental problem with public key encryption : suppose Bob wants to send an encrypted message to Alice, how does Bob get Alice's public key (assuming they haven't met before)? We can imagine there is a 'secure' central database where Alice's public key $\mathsf{pk}_{\text{Alice}}$ is stored. By a secure database, we mean that no adversary can alter the entries of this database. When Bob requests the database for Alice's public key, and the database sends $\mathsf{pk}_{\text{Alice}}$, an adversary can intercept this communication, and replace $\mathsf{pk}_{\text{Alice}}$ with pk_{Adv}. This completely compromises security, as the adversary has the secret key corresponding to pk_{Adv}, and can therefore read Bob's message. To handle this, we require a different cryptographic primitive - digital signatures. Digital signatures have immense applications in ensuring authenticity of digital communications.

8.1 *Digital Signatures: Definitions*

A digital signature scheme consists of three algorithms: $\mathsf{Setup}, \mathsf{Sign}$ and Verify. These algorithms have the following syntax:

- $\mathsf{Setup}\left(1^\lambda\right)$: The key generation algorithm takes as input the security parameter λ. It outputs a signing key sk and a verification key vk. In practice, the signing key sk must be kept secret, while the verification key vk is public.
- $\mathsf{Sign}\left(\mathsf{sk}, m\right)$: The signing algorithm takes as input the message m, the signing key sk and outputs a signature σ. The signing algorithm can be deterministic or randomized.
- $\mathsf{Verify}\left(\mathsf{vk}, m, \sigma\right)$: The verification algorithm takes as input the message m, signature σ and verification key vk. It outputs either 1 (indicating that σ is a valid signature for m) or 0 (indicating that σ is not a valid signature for m).

For correctness, we require that if $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}\left(1^\lambda\right)$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, then $\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$.

Security definitions for signature schemes

The security definition(s) are similar to the security definitions we had for MACs. An adversary can query for signatures on messages of its choice, and must finally output a new valid signature. If this new signature is on a new message, then we say that the adversary breaks *weak* unforgeability. If this new signature is on one of the queried messages, we say that the adversary breaks *strong* unforgeability. The formal security game is given below.

Unforgeability under Chosen Message Attack

- (Setup) Challenger chooses $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}\left(1^\lambda\right)$ and sends vk to the adversary.
- (Signature Queries) The adversary is allowed polynomially many signature queries. For every query m_i, the challenger sends $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$.
- (Forgery) Finally, the adversary sends a message m^* and signature σ^*. The adversary breaks **weak unforgeability** if $m^* \neq m_i$ for all i, and $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$. The adversary breaks **strong unforgeability** if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i, and $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$.

Figure 26: Unforgeability of Signature Schemes

Definition 8.1. *A signature scheme $\mathcal{S} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ satisfies weak (resp. strong) UnForgeability under Chosen Message Attacks if for any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{UFCMA}_{\text{weak/strong}}[\mathcal{A}, \mathcal{S}] = \mathsf{Pr}\left[\begin{array}{c}\mathcal{A} \text{ wins the weak (resp. strong)} \\ \text{unforgeability game (Fig. 26) against } \mathcal{S}\end{array}\right]$$

is negligible. ◊

Note: The adversary is allowed to repeat signing queries. Next, note that weak unforgeability guarantees strong unforgeability only if, for every message m, there is a unique signature that verifies. If the signature scheme is deterministic (but does not satisfy this uniqueness property), then we cannot conlude that weak unforgeability implies strong unforgeability.

8.2 *A few candidates discussed in class*

In this section, we discuss a few candidate signature schemes discussed in class.

Construction 8.2 (Textbook RSA signature scheme)**.**

- $\mathsf{Setup}\left(1^\lambda\right)$*: The key generation scheme chooses two n bit primes p and q. It sets $N = p \cdot q$. Next, it chooses an exponent $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$, together with $d \in \mathbb{Z}_{\phi(N)}$ such that $e \cdot d = 1 \bmod \phi(N)$. It sets the signing key $\mathsf{sk} = (N, d)$, and the verification key $\mathsf{vk} = (N, e)$.*
- $\mathsf{Sign}\,(\mathsf{sk} = (N, d), m \in \mathbb{Z}_N)$*: The signing algorithm outputs $\sigma = m^d \bmod N$.*
- $\mathsf{Verify}\,(\mathsf{vk} = (N, e), m, \sigma))$*: The verification algorithm outputs 1 iff $\sigma^e \bmod N = m$.*

First, note that the scheme satisfies correctness. It also satisfies the unique signatures property. However, the scheme is not secure. Given a signature on message m_1 and m_2, one can compute a signature on $m_1 \cdot m_2$. ◊

Construction 8.3 (Heuristic RSA based Candidate 1)**.** *Let H be a hash function (with appropriate domain and co-domain). This candidate is inspired by the RSA based encryption scheme.*

- $\mathsf{Setup}\left(1^\lambda\right)$*: The key generation scheme chooses two n bit primes p and q. It sets $N = p \cdot q$. Next, it chooses an exponent $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$, together with $d \in \mathbb{Z}_{\phi(N)}$ such that $e \cdot d = 1 \bmod \phi(N)$. It sets the signing key $\mathsf{sk} = (N, d)$, and the verification key $\mathsf{vk} = (N, e)$.*
- $\mathsf{Sign}\left(\mathsf{sk} = (N, d), m\right)$*: It samples $r \leftarrow \mathbb{Z}_N$, and outputs signature $\sigma = \left(r^d \bmod N, H(r) \oplus m\right)$.*
- $\mathsf{Verify}\left(\mathsf{vk} = (N, e), m, \sigma = (\sigma_1, \sigma_2)\right)$*: The verification algorithm computes $r = \sigma_1^e \bmod N$, and 1 iff $H(r) \oplus \sigma_2 = m$.*

This scheme is not secure; given a signature (σ_1, σ_2) on m, an adversary can produce a signature on any message m' by outputting $(\sigma_1, \sigma_2 \oplus m \oplus m')$. ◊

Construction 8.4 (Heuristic RSA based Candidate 2)**.** *This candidate is similar to Candidate 8.3, except that the second component of the signature is modified to avoid malleability. Instead of outputting $H(r) \oplus m$, the signing algorithm outputs $H(r \oplus m)$.*

- $\mathsf{Setup}\left(1^\lambda\right)$*: The key generation scheme chooses two n bit primes p and q. It sets $N = p \cdot q$. Next, it chooses an exponent $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$, together with $d \in \mathbb{Z}_{\phi(N)}$ such that $e \cdot d = 1 \bmod \phi(N)$. It sets the signing key $\mathsf{sk} = (N, d)$, and the verification key $\mathsf{vk} = (N, e)$.*
- $\mathsf{Sign}\left(\mathsf{sk} = (N, d), m\right)$*: It samples $r \leftarrow \mathbb{Z}_N$, and outputs signature $\sigma = \left(r^d \bmod N, H(r \oplus m)\right)$.*
- $\mathsf{Verify}\left(\mathsf{vk} = (N, e), m, \sigma = (\sigma_1, \sigma_2)\right)$*: The verification algorithm computes $r = \sigma_1^e \bmod N$, and 1 iff $H(r \oplus m) = \sigma_2$.*

This scheme is not secure; given a signature (σ_1, σ_2) on m, an adversary can produce a signature on any message m' computing $r = \sigma_1^e \bmod N$, and then outputting $(\sigma_1, H(r \oplus m'))$ (recall that the hash function H is publicly known). ◊

Construction 8.5 (Heuristic RSA based Candidate 3)**.** *This candidate is similar to Candidate 8.2, except that we compute a hash of m^d to avoid malleability.*

- $\mathsf{Setup}\left(1^\lambda\right)$*: The key generation scheme chooses two n bit primes p and*

q. It sets $N = p \cdot q$. Next, it chooses an exponent $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$, together with $d \in \mathbb{Z}_{\phi(N)}$ such that $e \cdot d = 1 \bmod \phi(N)$. It sets the signing key $\mathsf{sk} = (N, d)$, and the verification key $\mathsf{vk} = (N, e)$.

- $\mathsf{Sign}\,(\mathsf{sk} = (N, d), m \in \mathbb{Z}_N)$*: It outputs* $\sigma = H\left(m^d \bmod N\right)$.

In this construction, it is not possible to verify the signature, since the hash function is not invertible. ◊

8.3 *A secure construction in the random oracle model*

The textbook RSA scheme can be modified to get a candidate that is provably secure in the random oracle model. The construction uses a hash function $H : \{0,1\}^* \to \mathbb{Z}_N$ (this will be modeled as a random oracle in the security proof).

Construction 8.6 (RSA Full Domain Hash (RSA-FDH) Signatures)**.**

- $\mathsf{Setup}\left(1^\lambda\right)$ *: The signing key* $\mathsf{sk} = (N, d)$*, and the verification key* $\mathsf{vk} = (N, e)$ *(as usual,* $e \cdot d = 1 \bmod \phi(N)$*).*
- $\mathsf{Sign}\,(\mathsf{sk} = (N, d), m)$*: The signing algorithm outputs* $\sigma = H(m)^d \bmod N$*.*
- $\mathsf{Verify}\,(\mathsf{vk} = (N, e), m, \sigma)$*: The verification algorithm outputs* 1 *iff* $\sigma^e \bmod N = H(m)$*.*

The signing algorithm is deterministic, and the signature scheme has unique signatures. It is also provably secure in the random oracle model, assuming the RSA assumption.
Note that the hash function maps the message string to a number in $\mathbb{Z}_N$. This will be important in the random oracle proof. Hence, this construction is often referred to as RSA signatures with full domain hash*.* ◊

Before we see the proof of Construction 8.6, let us discuss how RSA based signatures are implemented in practice, and some associated vulnerabilities. As mentioned above, it is important that the co-domain of H is $\mathbb{Z}_N$. However, in practice, most hash functions map input strings to 128 or 256 bits. In the 90s, the PKCS standard specified the following approach for signing (arbitrary length) messages.

Construction 8.7 (PKCS Signatures)**.**

- $\mathsf{Setup}\left(1^\lambda\right)$ *: The signing key* $\mathsf{sk} = (N, d)$*, and the verification key* $\mathsf{vk} = (N, e)$ *(as usual,* $e \cdot d = 1 \bmod \phi(N)$*). Here, N is a* 1024 *bit modulus.*
- $\mathsf{Sign}\,(\mathsf{sk} = (N, d), m)$*: The signing algorithm first computes* $h = H(m)$*, which is a* 128 *bit string. Next, it pads the string h to express it as a*

number in $\mathbb{Z}_N$. The padding string is first computed as follows:

$$x = \underbrace{\texttt{0x00}}_{1\ byte}\ \underbrace{\texttt{0x01}}_{1\ byte}\ \underbrace{\texttt{0xff 0xff ... 0xff}}_{109\ bytes}\ \underbrace{\texttt{0x00}}_{1\ byte}\ \underbrace{h}_{16\ bytes}$$

This string is then expressed as a number in $\mathbb{Z}_N$, and then the signing algorithm outputs x^d.

- Verify $(\mathsf{vk} = (N, e), m, \sigma)$*: The verification algorithm computes* $x = \sigma^e \bmod N$*, and expresses it as a sequence of* 128 *bytes. It then checks the following:*
  - *the first two bytes are* 0x00 *and* 0x01 *respectively*
  - *the next* 109 *bytes are* 0xff
  - *the next byte is* 0x00
  - *the last* 16 *bytes are equal to* $H(m)$.

$\Diamond$

In practice, developers optimized the PKCS scheme to allow fast signing/verification. One popular optimization was to set $e = 3$, and ensuring the modulus N is chosen such that $\phi(N)$ is co-prime to 3. This optimization ensures that verification is faster, but can potentially lead to security attacks.

Certain implementations (including one that was used by Firefox in the early 2000s) optimized the verification further. In these implementations, they did not check that there are 109 bytes of 0xff. Instead, they just checked that the first two bytes are 0x00 and 0x01 respectively, the next few bytes are 0xff, the byte after the string of 0xff is 0x00. They took the next 16 bytes, and checked if this is equal to $H(m)$.

As it turns out, this aggressive optimization leads to a fatal forgeability attack, and it is possible to compute a valid signature for *any* message m! This was shown by David Bleichenbacher in 2006, and we describe the attack below. The moral of this episode is that we must avoid using a small modulus e.

One can argue that the 'real' issue here is that the implementation did not check that there are 109 *bytes of* 0xff*. Unfortunately, such implementation mistakes often happen in practice, and a simple fix is to completely avoid small exponent e or small private exponent d.*

A valid forgery for message m is a 128 bit string σ such that

$$\sigma^3 \bmod N = \underbrace{\texttt{0x00}}_{1\ \text{byte}}\ \underbrace{\texttt{0x01}}_{1\ \text{byte}}\ \underbrace{\texttt{0xff}}_{1\ \text{byte}}\ \underbrace{\texttt{0x00}}_{1\ \text{byte}}\ \underbrace{H(m)}_{16\ \text{bytes}}\ \underbrace{\text{any garbage bytes}}_{108\ \text{bytes}}$$

Consider the following numbers in $\mathbb{Z}_N$:

$$\alpha = \texttt{0x00 0x01 0xff 0x00}\ \underbrace{H(m)}_{16\ \text{bytes}}\ \underbrace{\texttt{0x00}\ldots\texttt{0x00}}_{108\ \text{bytes}}$$

$$\beta = \texttt{0x00 0x01 0xff 0x00}\ \underbrace{H(m)}_{16\ \text{bytes}}\ \underbrace{\texttt{0xff}\ldots\texttt{0xff}}_{108\ \text{bytes}}$$

If we find a cubic integer θ in the range $[\alpha, \beta]$, then we can set $\sigma = \theta^{1/3}$ (there is no $\bmod N$ when we compute the cube root of θ). Note that $\theta = \sigma^3$ (this equality

holds without taking $\bmod N$, and therefore also holds when we take $\bmod N$). A natural candidate for finding a cube in this range is to consider $\theta = \left\lceil \alpha^{1/3} \right\rceil^3$. All that remains to show is $\theta \leq \beta$. Let $z = \alpha^{1/3}$ (this may not be an integer). Note that $\alpha = z^3 \leq \theta < (z+1)^3$. Also, $z < 255^{128/3} < 255^{43}$ (because z^3 can be expressed using 128 bytes).

$$\begin{aligned}\theta &< \alpha + 3z^2 + 3z + 1 \\ &< \alpha + 6z^2 \\ &< \alpha + 6 \cdot 255^{86} \\ &< \beta\end{aligned}$$

The last inequality follows because $\beta - \alpha = 255^{108}$.

> **(*) Exercise 8.1.** *Would Bleichenbacher's attack work if the public exponent was set to be $e = 7$? Would it work if we used a hash function H that maps the messages to 64 byte (that is, 512 bit) strings?*

Proof of Construction 8.6:

Lecture 26: November 10th, 2023

We will prove security of Construction 8.6 in the random oracle model. First, let us recall the unforgeability security game in the random oracle model. It is similar to the game described in Figure 26. However, the adversary is now allowed queries to the random oracle.

- (Setup) Challenger chooses $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^n)$ and sends vk to the adversary. It also chooses a uniformly random function f that maps $\{0,1\}^*$ to $\mathbb{Z}_N$.
- (Queries) The adversary is allowed two kinds of queries: random oracle queries and signature queries.
  - (Signature Queries) The adversary is allowed polynomially many signature queries. For every query m_i, the challenger sends $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$. Note that the signing algorithm can query the random oracle too.
  - (Random oracle queries) The adversary queries for the random oracle output on x_i, and gets $f(x_i)$.
- (Forgery) Finally, the adversary sends a message m^* and signature σ^*. The adversary breaks **weak unforgeability** if $m^* \neq m_i$ for all i, and $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$. Note that the verification algorithm (and hence the winning condition also) can depend on the random oracle. Also, while m^* should not be equal to any of the m_i strings, it is allowed to be equal to one of the random oracle queries.

Lemma 8.8. *Assuming the RSA assumption holds, Construction 8.6 is a digital signature scheme that is strongly unforgeable in the random oracle model.*

Proof idea: Consider the following simplified security game: the adversary sends the target message m^* (that is, the one for which it will produce a forgery) at the start of the game. Next, it queries for polynomially many random oracle queries. After all the random oracle queries, it queries for a signature on one of the random oracle queries, say x_i. Finally outputs the forgery corresponding to m^*.

As usual, we will start with this security game, then gradually modify the game until we reach a point where we are sure that the adversary cannot produce a forgery. One (or more) of these steps will rely on the security of RSA. The reduction breaking RSA will need to first give out responses to random oracle queries. After all the random oracle queries are answered, it needs to produce a signature on x_i. Let y_i be the random oracle response corresponding to x_i. The reduction needs to compute the e^{th} root of y_i. How does the reduction compute the e^{th} root of y_i without knowing the inverse of e?

The key idea here is the following: when the random oracle queries are handled, instead of picking a uniformly random y_i and sending this as the response, sample a uniformly random z_i, set $y_i = z_i^e$ and store (x_i, y_i, z_i) in the table $\mathcal{T}$. The distribution of e^{th} power of a uniformly random element of $\mathbb{Z}_N$ is identical the distribution of a uniformly random element in $\mathbb{Z}_N$, and hence y_i is sent from the correct distribution. Finally, when it receives a signature query for x_i, the reduction can send z_i as the signature.

Proof. We will prove security of our RSA based candidate via a sequence of games. The first game is identical to our security game in the ROM. The challenger, instead of choosing the random function, samples the function on-the-fly.

Game 0: This is the original unforgeability game in the random oracle model.

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Queries)
  - (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random number $y_i \leftarrow \mathbb{Z}_N$, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i) \in \mathcal{T}$, the challenger sends $y_i^d \bmod N$ to the adversary. Else, it samples a fresh $y_i \leftarrow \mathbb{Z}_N$, adds (m_i, y_i) to $\mathcal{T}$, and sends $\sigma_i = y_i^d \bmod N$ as the signature.
- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:
  - $m^* \neq m_i$ for all i
  - If $(m^*, y^*) \notin \mathcal{T}$, the challenger samples a random string y^*, and adversary wins if $(\sigma^*)^e = y^*$. If $(m^*, y^*) \in \mathcal{T}$, the adversary wins if $(\sigma^*)^e = y^*$.

Game 1: This game is identical to the previous game, except that the winning condition is altered slightly. In order to win, the adversary must have queried the random oracle at the forgery message.

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Queries)
  - (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random number $y_i \leftarrow \mathbb{Z}_N$, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i) \in \mathcal{T}$, the challenger sends y_i^d mod N to the adversary. Else, it samples a fresh $y_i \leftarrow \mathbb{Z}_N$, adds (m_i, y_i) to $\mathcal{T}$, and sends $\sigma_i = y_i^d$ as the signature.
- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:
  - $m^* \neq m_i$ for all i
  - If $(m^*, y^*) \notin \mathcal{T}$, the challenger samples a random string y^* and adversary wins if $(\sigma^*)^e = y^*$.
    $(m^*, y^*) \in \mathcal{T}$, and $(\sigma^*)^e = y^*$.

Game 2: In this game, the challenger guesses the random oracle query that corresponds to the forgery. Let Q denote the number of random oracle queries made by the adversary. The adversary wins if the guess is correct.

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
  The challenger chooses an index $i^* \leftarrow [Q]$.
- (Queries)
  - (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random number $y_i \leftarrow \mathbb{Z}_N$, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i) \in \mathcal{T}$, the challenger sends y_i^d mod N to the adversary. Else, it samples a fresh $y_i \leftarrow \mathbb{Z}_N$, adds (m_i, y_i) to $\mathcal{T}$, and sends $\sigma_i = y_i^d$ as the signature.
- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:
  - $m^* \neq m_i$ for all i
  - m^* is the $(i^*)^{\text{th}}$ random oracle query.

– $(m^*, y^*) \in \mathcal{T}$, and $(\sigma^*)^e = y^*$.

Game 3: In this game, the challenger does not use d. Instead, it samples the y_i strings differently (for responding to new random oracle/signature queries). For each fresh random oracle query x_i, it first samples $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$ and adds (x_i, y_i, z_i) to the table. It performs a similar thing for signing queries. Note that $z_i = y_i^d \bmod N$.

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.

  The challenger chooses an index $i^* \leftarrow [Q]$.

- (Queries)

  – (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i, z_i) \in \mathcal{T}$, the challenger sends y_i in response.
  Else, it chooses a uniformly random number $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (x_i, y_i, z_i) to $\mathcal{T}$, and sends y_i to the adversary.

  – (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i, z_i) \in \mathcal{T}$, the challenger sends z_i to the adversary.
  Else, it samples a fresh $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (m_i, y_i, z_i) to $\mathcal{T}$, and sends $\sigma_i = z_i$ as the signature.

- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:

  – $m^* \neq m_i$ for all i
  – m^* is the $(i^*)^{\text{th}}$ random oracle query.
  – $(m^*, y^*) \in \mathcal{T}$, and $(\sigma^*)^e = y^*$.

Analysis: First, observe that the only difference between Game 0 and Game 1 is that in Game 0, the adversary can win even if it has not queried the random oracle on m^*. However, in this case, y^* is chosen at random (after the adversary sends its forgery), and therefore, the adversary's winning probability in this case is at most $1/N$. Hence the winning probabilities in Game 0 and Game 1 differ by at most $1/N$, which is negligible.

Next, note that if an adversary wins with probability ϵ in Game 1, then it wins with probability ϵ/Q in Game 2. This is because the challenger's guess i^* is correct with probability $1/Q$. Conditioned on the guess being correct, the adversary's winning probability is still ϵ, and therefore the winning probability in Game 2 is ϵ/Q.

Third, note that Game 2 and Game 3 are almost identical from the adversary's perspective. This is because sampling $y_i \leftarrow \mathbb{Z}_N$ is identical to sampling $z_i \leftarrow \mathbb{Z}_N$ and setting $y_i = z_i^e$.

Finally, we need to show that any p.p.t. adversary has negligible advantage in Game 3. This follows from the RSA assumption. We include the complete reduction below.

Claim 8.9. *Suppose there exists a p.p.t. adversary $\mathcal{A}$ that makes Q random oracle queries, and wins in Game 3 with probability ϵ. Then there exists a reduction algorithm $\mathcal{B}$ that breaks the RSA assumption with probability ϵ.*

Proof. The reduction algorithm receives (N, e, y^*) from the RSA challenger. It sets $\mathsf{vk} = (N, e)$ and sends it to $\mathcal{A}$. It also maintains an empty table $\mathcal{T}$ and picks $i^* \leftarrow [Q]$. The adversary makes polynomially many signing queries, and Q random oracle queries. For each of the queries, the reduction follows the Game 3 challenger, and does the following:

- (Signing query) The reduction, on receiving a query for m_i, checks if $(m_i, y_1, z_i) \in \mathcal{T}$. If so, it sends z_i. Else, it samples $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (m_i, y_i, z_i) to $\mathcal{T}$ and sends z_i.
- (Random oracle query) The reduction, on receiving a random query for x_i, checks if $(x_i, y_i, z_i) \in \mathcal{T}$. If so, it sends y_i.

  Else, if it is the $(i^*)^{\text{th}}$ random oracle query, the challenger sends y^* (and adds nothing to the table $\mathcal{T}$).

  Else, it samples $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (x_i, y_i, z_i) to $\mathcal{T}$ and sends y_i.

Finally, the adversary sends a forgery (m^*, σ^*). If m^* is not equal to the $(i^*)^{\text{th}}$ random oracle query, then the adversary loses Game 3. If it is equal to the $(i^*)^{\text{th}}$ random oracle query, and verification passes, then $(\sigma^*)^e = y^*$. Note, in order to win in Game 3, the adversary does not query for signature on m^* (and therefore the reduction does not need to add m^* to $\mathcal{T}$). □

Putting everything together, we get that if there exists an adversary that breaks the security of Construction 8.6 with probability ϵ, then there exists a p.p.t. reduction that breaks RSA security with probability ϵ/Q. □

(*) Exercise 8.2. *Suppose we consider a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ (for instance, in practice, the hash function's output is $\{0,1\}^{128}$ or $\{0,1\}^{256}$). If we use Construction 8.6 with this hash function, we will not get a security proof in the random oracle model. Mofidy the RSA assumption to get a computational problem that would suffice for proving security of this construction in the random oracle model.*

Lecture 27: November 14th, 2023

The above result can be improved to get better concrete bounds on the reduction's winning probability. Suppose an adversary makes Q_{ro} random oracle calls and Q_{sig} signature queries during the lifetime of a signature scheme. Typically, Q_{sig} is quite small (≈ 1000). However, Q_{ro} represents the number of evaluations of the hash function that can be performed during the lifetime of the signature scheme, and this can be quite large ($\approx 10^{30}$). In Lemma 8.8, we showed that the reduction algorithm wins the RSA game with probability ϵ/Q_{ro}. Using a better reduction algorithm (and without changing the construction), the reduction's success probability can be improved to $\omega(\epsilon/Q_{\text{sig}})$.

To see this, let us consider a simplified security game where the adversary first makes Q random oracle queries, then makes a signature query for the last random oracle query, and finally submits a forgery on one of the first $Q-1$

random oracle queries. Suppose an adversary wins this simplified security game with probability ϵ. Can we build a reduction algorithm that breaks RSA with probability ϵ? If we try to guess the random oracle query that corresponds to the forgery, then we lose a factor of $(Q-1)$.

Instead, the reduction algorithm does the following: it receives (N, e, y) from the RSA challenger, and sends (N, e) as the verification key. Next, it must respond to random oracle queries. For the i^{th} random oracle query x_i, if $i \leq Q-1$, it picks a uniformly random $c_i \leftarrow \mathbb{Z}_N^*$, sends $c_i^e \cdot y \bmod N$ as the random oracle response. For the Q^{th} random oracle query x_Q, it picks a uniformly random $c_Q \leftarrow \mathbb{Z}_N^*$ and sends $c_Q^e \bmod N$. Note that the signature on x_Q is c_Q. Finally, suppose the adversary submits a valid forgery σ^* on x_i for some $i \in [Q-1]$. Then, $(\sigma^*)^e \bmod N = c_i^e \cdot y \bmod N$, and therefore the e^{th} root of y is $\sigma^* \cdot (c_i)^{-1} \bmod N$. As a result, the reduction succeeds with probablity ϵ.

Note that in the above simplified security game, the reduction knew which random oracle queries will correspond to signature queries, and which ones will contain the forgery. In the general security game, the reduction does not know this, and therefore must guess. For each query, the reduction does the following: with probability p, it guesses that there will be a signature query corresponding to this random oracle query (and as a result, it will set the random oracle query in such a manner that it can answer the signature query), and with probability $1-p$, it guesses that it will not need to answer a signature query for this string (and therefore, it uses the RSA challenge for generating the random oracle response). See Exercise 8.3.

Exercise 8.3. *In this exercise, we will construct a better RSA reduction for Claim 8.9. The reduction algorithm receives the RSA challenge (N, e, y), and sends $\mathsf{vk} = (N, e)$. For each random oracle query, the reduction, with probability p, guesses that this will be a signature query; it chooses c_i and sends $c_i^e \bmod N$ as the random oracle response. For the other queries, it chooses c_i and sends $c_i^e \cdot y \bmod N$ as the random oracle response.*

Complete the reduction (that is, specify the value of p) and compute the success probability of the reduction algorithm.

8.4 *Secure Signature Schemes in the Standard Model*

In the last section, we saw a signature scheme whose security was proven in the random oracle model. Here, we will discuss a signature scheme whose security can be proven in the standard model. The construction uses the most basic cryptographic primitives - one way functions and collision resistant hashing. We will start with a signature scheme that is one-time secure.

One-time secure digital signatures

Let $f : \{0,1\}^\ell \to \{0,1\}^\ell$ be a secure one-way function (think of discrete log, RSA, etc). First, we will consider a signature scheme with message space $\{0,1\}^n$, whose security relies on the one-wayness of f.

Construction 8.10 (Lamport's one-time signatures)**.**

- $\mathsf{Setup}(1^n)$: *Choose* $2n$ *strings* $a_{i,b} \leftarrow \{0,1\}^\ell$ *for each* $i \in [n]$, $b \in \{0,1\}$. *The secret key is* $\mathsf{sk} = \left(a_{i,b}\right)_{i\in[n],b\in\{0,1\}}$ *and the verification key is* $\mathsf{vk} = \left(f(a_{i,b})\right)_{i\in[n],b\in\{0,1\}}$.
- $\mathsf{Sign}\left(\mathsf{sk} = \left(a_{i,b}\right)_{i,b}, m = (m_1,\ldots,m_n)\right)$: *The signing algorithm outputs* $\sigma = \left(a_{i,m_i}\right)_{i\in[n]}$ *as the signature.*
- $\mathsf{Verify}\left(\mathsf{vk} = \left(\mathsf{vk}_{i,b}\right)_{i,b}, m = (m_1,\ldots,m_n), \sigma = (z_i)_i\right)$: *The verification algorithm checks if* $f(z_i) = \mathsf{vk}_{i,m_i}$ *for all* $i \in [n]$. *If so, it outputs* 1, *else it outputs* 0.

$\Diamond$

First, check that correctness holds. The scheme is deterministic.

(*) Exercise 8.4. *Is Construction 8.10 a scheme with unique signatures? Would this be a scheme with unique signatures if Discrete Log (with prime order groups)/RSA are used?*

Construction 8.10 is one-time **weakly unforgeable**. Intuitively, this is because the adversary queries for a signature on m, and must send a forgery on a different message m^*. Since these two strings differ in at least one position, the reduction can guess this position, and plant the OWF challenge at this position. The formal reduction is given below.

Claim 8.11. *Assuming f is a secure one way function, the construction described above is a weakly unforgeable one-time secure signature scheme.*

Proof. The reduction algorithm receives y from the OWF challenger. It guesses an index $i^* \leftarrow [n]$ and a bit $b^* \leftarrow \{0,1\}$. For all $(i,b) \neq (i^*,b^*)$, it chooses $a_{i,b}$ and sets $\mathsf{vk}_{i,b} = f(a_{i,b})$. It sets $\mathsf{vk}_{i^*,b^*} = y$ and sends vk to the adversary. The adversary sends a message $m \in \{0,1\}^n$. If $m_{i^*} = b^*$, then the reduction algorithm quits. Else, it sends $\sigma = \left(a_{i,m_i}\right)_i$ as the signature. Finally the adversary outputs a forgery (m^*,σ^*). If $m^* \neq m$ and $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$, the reduction checks if $m^*_{i^*} = b^*$. If so, it sends $x = \sigma^*_{i^*}$ to the OWF challenger. Note that since the forgery verifies, if $m^*_{i^*} = b^*$, then $f(x) = y$. If the adversary breaks weak unforgeability with probability ϵ, then the reduction breaks the OWF security with probability $O(\epsilon/n)$. □

(*) Exercise 8.5. *Construction 8.10 may not be strongly unforgeable. Let f be a secure one-way function. Construct a new one-way function f' such that f' is also a secure one way function, but when used in Construction 8.10, the resulting signature scheme is not one-time strongly unforgeable.*

One-time unforgeable signatures with unbounded message space: Given a one-time unforgeable signature scheme with bounded message space, we can combine it with a collision resistant hash function to get a one-time signature scheme with unbounded messages, satisfying one-time security. Simply hash the message and compute a signature on the digest. The hash-and-sign approach also works for one-time security.

Construction 8.12 (Lamport's one-time signatures, large message space)**.** *Let $\mathcal{H} = \{ H_k : \{0,1\}^* \to \{0,1\}^n \}_{k \in \mathcal{K}}$ be a family of collision resistant hash functions with key space $\mathcal{K}$, and let f be a one-way function.*

- $\mathsf{Setup}(1^n)$: *Choose $2n$ strings $a_{i,b} \leftarrow \{0,1\}^\ell$ for each $i \in [n]$, $b \in \{0,1\}$ and hash key $k \leftarrow \mathcal{K}$. The secret key is $\mathsf{sk} = \left(k, \left(a_{i,b}\right)_{i \in [n], b \in \{0,1\}}\right)$ and the verification key is $\mathsf{vk} = \left(k, \left(f(a_{i,b})\right)_{i \in [n], b \in \{0,1\}}\right)$.*
- $\mathsf{Sign}\left(\mathsf{sk} = \left(k, \left(a_{i,b}\right)_{i,b}\right), m\right)$: *The signing algorithm computes $h = H_k(m)$, and outputs $\sigma = \left(a_{i,h_i}\right)_{i \in [n]}$ as the signature.*
- $\mathsf{Verify}\left(\mathsf{vk} = \left(k, \left(\mathsf{vk}_{i,b}\right)_{i,b}\right), m, \sigma = (z_i)_i\right)$: *The verification algorithm computes $h = H_k(m)$, checks if $f(z_i) = \mathsf{vk}_{i,h_i}$ for all $i \in [n]$. If so, it outputs 1, else it outputs 0.*

$\Diamond$

The weak unforgeability follows from collision-resistance of $\mathcal{H}$ and the one-wayness of f. Either the adversary produces a query message m and forgery that hash to the same string (in which case we have a collision for $\mathcal{H}$), or we have an attack on the one-wayness of f (similar to proof of Claim 8.11).

From one-time security to unbounded security with stateful signing

Construction 8.10 is not even two-time secure. An adversary can query for 0^n, followed by 1^n, and it would learn the entire secret key.

Let $\mathcal{S}_{\text{ot}} = (\mathsf{Setup}_{\text{ot}}, \mathsf{Sign}_{\text{ot}}, \mathsf{Verify}_{\text{ot}})$ be a one-time secure signature scheme with message space $\{0,1\}^*$. Our aim in this section is to construct a signature scheme with unbounded security. As a first step, we will construct a signature scheme with the following properties:

- the message space is $\{0,1\}^*$.
- the signing algorithm is stateful. After each signing query, the signing key is updated for the next signature. However, the verification key does not get updated. This can be useful in practical settings where we have a single centralized signing party. Note that it is important that verification is stateless, since the verification can be performed by any party that possesses the verification key (and in practice, it is infeasible to update publicly available keys).
- we will allow the signature size, as well as the signing key's size, to grow with each query. At the end of this section (in Exercise **??**, we will discuss

how to ensure that the signature size does not grow with the number of queries.

The following candidates were discussed in class. First, if we have an apriori bound B on the maximum number of signature queries, then we can sample B signing/verification keys, and use each key exactly once.

Construction 8.13 (From One-Time Signatures to B-Time Stateful Signatures)**.** *Let $\mathcal{S}_{\mathsf{ot}} = (\mathsf{Setup}_{\mathsf{ot}}, \mathsf{Sign}_{\mathsf{ot}}, \mathsf{Verify}_{\mathsf{ot}})$ be a one-time secure signature scheme with message space $\{0,1\}^*$.*

- $\mathsf{Setup}\left(1^\lambda\right)$: *The setup algorithm chooses $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Setup}_{\mathsf{ot}}\left(1^\lambda\right)$ for each $i \in [B]$. It sets $\mathsf{sk} = \left(\mathsf{st} = 0, (\mathsf{sk}_i)_{i\in[B]}\right)$ and $\mathsf{vk} = (\mathsf{vk}_i)_{i\in[B]}$.*
- $\mathsf{Sign}\left(\mathsf{sk} = \left(\mathsf{st}, (\mathsf{sk}_i)_{i\in[B]}\right), m\right)$*: The signing algorithm increments* st *by* 1*, and outputs* $(\mathsf{st}, \mathsf{Sign}_{\mathsf{ot}}(\mathsf{sk}_{\mathsf{st}}, m))$*. The updated signing key is* $\left(\mathsf{st}, (\mathsf{sk}_i)_{i\in[B]}\right)$.
- $\mathsf{Verify}\left(\mathsf{vk} = (\mathsf{vk}_i)_{i\in[B]}, m, (\mathsf{st}, \sigma)\right)$: *The verification algorithm outputs* $\mathsf{Verify}_{\mathsf{ot}}(\mathsf{vk}_{\mathsf{st}}, m, \sigma)$.

Note that since this is a stateful signature scheme, each one-time signing key is used at most once. Hence the scheme is B-time secure, assuming $\mathcal{S}_{\mathsf{ot}}$ is one-time secure. ◊

We would like to handle unbounded signature queries. Let us consider the following candidate proposed in class.

Construction 8.14 (From One-Time Signatures to Unbounded Stateful Signatures: Attempt 1)**.** *The key idea in this construction is to choose a fresh signing/verification key each time a new message is signed. However, this new signing/verification key must also be 'certified', and this is done using the previous signing key.*

$\mathsf{Setup}\left(1^\lambda\right)$: *The setup algorithm chooses* $(\mathsf{sk}_0, \mathsf{vk}_0) \leftarrow \mathsf{Setup}_{\mathsf{ot}}\left(1^\lambda\right)$*. It sets* $\mathsf{sk} = (0, \mathsf{sk}_0)$ *and* $\mathsf{vk} = \mathsf{vk}_0$.

$\mathsf{Sign}\left(\mathsf{sk} = \left(t, \mathsf{sk}_0, (\mathsf{sk}_i)_{i\in[t]}, (\mathsf{vk}_i)_{i\in[t]}, (\sigma_i)_{i\in[t]}\right), m\right)$*: The signing key consists of* $t \in \mathbb{N}$ *which denotes the number of signatures issued so far, the starting signing key* sk_0*, the t signing/verification keys chosen during the previous t signings, as well as t signatures (these will be signatures on the verification keys). The signing algorithm chooses* $(\mathsf{sk}_{t+1}, \mathsf{vk}_{t+1}) \leftarrow \mathsf{Setup}_{\mathsf{ot}}\left(1^\lambda\right)$*, computes* $\sigma_{t+1} \leftarrow \mathsf{Sign}_{\mathsf{ot}}\left(\mathsf{sk}_t, \mathsf{vk}_{t+1}\right)$ *and* $\sigma = \mathsf{Sign}_{\mathsf{ot}}\left(\mathsf{sk}_{t+1}, m\right)$.
The signature consists of σ, together with $(\mathsf{vk}_i)_{i\in[t+1]}$ *and* $(\sigma_i)_{i\in[t+1]}$.
The updated secret key is $\left(t+1, \mathsf{sk}_0, (\mathsf{sk}_i)_{i\in[t+1]}, (\mathsf{vk}_i)_{i\in[t+1]}, (\sigma_i)_{i\in[t+1]}\right)$.

$\mathsf{Verify}\left(\mathsf{vk}_0, m, \left(\sigma, (\mathsf{vk}_i)_{i\in[t+1]}, (\sigma_i)_{i\in[t+1]}\right)\right)$: *The verification algorithm performs the following checks:*

- $\mathsf{Verify}_{\mathrm{ot}}(\mathsf{vk}_{t+1}, m, \sigma) = 1$.
- *For all* $i \in [t+1]$, $\mathsf{Verify}_{\mathrm{ot}}(\mathsf{vk}_{i-1}, \mathsf{vk}_i, \sigma_i) = 1$. *It outputs* 1 *only if all the above checks pass.*

The above scheme satisfies correctness. However, we cannot prove security of this candidate using one-time security of $\mathcal{S}_{\mathrm{ot}}$. *Note that* sk_1 *is used to sign the first message, as well as the second verification key* vk_2. *As suggested in class, one way to fix this is to use a two-time secure signature scheme (and build the two-time scheme using Construction 8.13). There is a more direct approach, which we discuss in Construction 8.15.* ◊

This above construction almost gives us what we want, except that each signing key is used twice. There is an easy fix here: choose the next signing/verification key, and sign the next verification key together with the message.

Construction 8.15 (From One-Time Signatures to Unbounded Stateful Signatures: Attempt 2)**.**

$\mathsf{Setup}\left(1^\lambda\right)$: *The setup algorithm chooses* $(\mathsf{sk}_0, \mathsf{vk}_0) \leftarrow \mathsf{Setup}_{\mathrm{ot}}\left(1^\lambda\right)$. *It sets* $\mathsf{sk} = (0, \mathsf{sk}_0)$ *and* $\mathsf{vk} = \mathsf{vk}_0$.

$\mathsf{Sign}\left(\mathsf{sk} = \left(t, \mathsf{sk}_0, (\mathsf{sk}_i, \mathsf{vk}_i, m_i, \sigma_i)_{i\in[t]}\right), m\right)$: *The signing key consists of* $t \in \mathbb{N}$ *which denotes the number of signatures issued so far, the starting signing key* sk_0, *the* t *signing/verification keys chosen during the previous* t *signings, the* t *messages signed so far, as well as* t *signatures. The signing algorithm chooses* $(\mathsf{sk}_{t+1}, \mathsf{vk}_{t+1}) \leftarrow \mathsf{Setup}_{\mathrm{ot}}\left(1^\lambda\right)$, *computes* $\sigma_{t+1} \leftarrow \mathsf{Sign}_{\mathrm{ot}}(\mathsf{sk}_t, (m \,||\, \mathsf{vk}_{t+1}))$. *The signature consists of* $(\mathsf{vk}_i, m_i, \sigma_i)_{i\in[t+1]}$.
The updated secret key is $\left(t+1, \mathsf{sk}_0, (\mathsf{sk}_i, \mathsf{vk}_i, m_i, \sigma_i)_{i\in[t+1]}\right)$, *where* $m_{t+1} = m$.

$\mathsf{Verify}\left(\mathsf{vk}_0, m, \left((\mathsf{vk}_i)_{i\in[t+1]}, (\mathsf{vk}_i)_{i\in[t+1]}, (\sigma_i)_{i\in[t+1]}\right)\right)$: *The verification algorithm performs the following checks:*

- $m_{t+1} = m$
- *For all* $i \in [t+1]$, $\mathsf{Verify}_{\mathrm{ot}}(\mathsf{vk}_{i-1}, (m_i \,||\, \mathsf{vk}_i), \sigma_i) = 1$. *It outputs* 1 *only if all the above checks pass.*

The above scheme satisfies correctness. Also, note that if t *signatures are issued, then each signing key is used to sign exactly one string. Proving security of this signature scheme requires some careful book-keeping.* ◊

In the above signature scheme, the signature size, as well as the secret key's size grows with each signature. Using a tree-based structure, we can ensure that the size of the signature and signing key is bounded by $\mathrm{poly}(n)$.

References

[Ble98] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[BS23] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2023.

[CS98] Ronald Cramer and Victor Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

[DDN03] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable Cryptography. *SIAM Rev.*, 45(4):727–784, 2003.

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions (Extended Abstract). In *FOCS*, pages 464–479, 1984.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A Pseudorandom Generator from any One-way Function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[LR85] Michael Luby and Charles Rackoff. How to Construct Pseudo-Random Permutations from Pseudo-Random Functions (Abstract). In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, page 447. Springer, 1985.

[NY90] Moni Naor and Moti Yung. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 427–437. ACM, 1990.

[RS91] Charles Rackoff and Daniel R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1991.

[RSA83] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems (Reprint). *Commun. ACM*, 26(1):96–99, 1983.

[Vau02] Serge Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer, 2002.

[Yao82] Andrew Chi-Chih Yao. Theory and Applications of Trapdoor Functions (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91. IEEE Computer Society, 1982.

A Probability Review

Many thanks to Abhinav Kumar for preparing this section.

A.1 *Basic Definitions*

- A **finite probability space**, denoted as (Ω, ω), consists of a finite set $\Omega = \{\omega_1, ..., \omega_n\}$ containing possible outcomes of a probabilistic event.
  A **probability mass function** $p : \Omega \to [0, 1]$ that assigns a probability value to each outcome ω in Ω such that $\sum_{\omega \in \Omega} p(\omega) = 1$.

  Example: Rolling two dice
  Probability space:
  The sample space will include all pairs of numbers from 1 to 6

  $$\Omega = \{(1,1), (1,2), ..., (1,6), (2,1), (2,2), ..., (2,6), ..., (6,1), (6,2), ..., (6,6)\}$$

  Probability mass function:

  $$p(\omega) = 1/36 \text{ for all } \omega \in \Omega$$

- An **event** E over a probability space (Ω, ω) is a set $E \subseteq \Omega$. The probability of event E, denoted by $P[E]$ is defined as

  $$P(E) = \sum_{\omega \in E} p(\omega)$$

  For an outcome $\omega \in \Omega$,

  $$P(\omega) = p(\omega)$$

  Example: For rolling two dice, let's define an event A as the sum of the numbers rolled on the two dice being equal to 7. The event A can be represented as follows:

  $$A = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\}$$

- A **random variable** X over a probability space (Ω, ω) is a real-valued function $X : \Omega \to \mathbb{R}$.

  Example: For rolling two dice, let's define a random variable X = "Sum of the numbers rolled on two dice".
  The possible values that X can take are the sums of the numbers from 2 to 12.

  Note For the remainder of this handout, we will assume all random variables are defined over a probability space (Ω, ω).

A.2 *Expected Value and Variance*

Expected Value

The expected value $E[X]$ of a random variable X is defined as

$$E[X] = \sum_{w \epsilon \Omega} X(\omega) P(\omega)$$

Example: Let's calculate the expected value for the above defined random variable X = "Sum of the numbers rolled on two dice".
We know that $p(\omega) = 1/36$ for all ω.

$$E[X] = (2 * 1/36) + (3 * 2/36) + (4 * 3/36) + (5 * 4/36) + (6 * 5/36) + (7 * 6/36)$$

$$+(8 * 5/36) + (9 * 4/36) + (10 * 3/36) + (11 * 2/36) + (12 * 1/36)$$

$$E[X] = 252/36 = 7$$

- **Linearity of expectation** For all random variables X, Y and all $\alpha, \beta \in \mathbb{R}$

  $$E[\alpha X + \beta Y] = \alpha E[X] + \beta E[Y]$$

  Example: For rolling two dice, Let's take two random variables Y and Z as
  Y = "The value rolled on the first die"
  Z = "The value rolled on the second die"
  We can see that $X = Y + Z$.
  Now,

  $$E[Y] = (1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5$$

  $$E[Z] = (1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5$$

  Using linearity of expectation:

  $$E[X] = E[Y] + E[Z] = 3.5 + 3.5 = 7$$

  This is the same as what we got in the above example directly.

Variance

The variance $Var(X)$ of a random variable X is defined as

$$Var(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2$$

Example: For rolling two dice, let's consider the random variable X defined above.
We know that, $E[X] = 7$
Similarly, we can calculate $E[X^2] \approx 54.833$

$$Var(X) = E[X^2] - E[X]^2$$

$$\approx 54.833 - 7^2 \approx 5.833$$

A.3 *Conditional Probability and Bayes Theorem*

Conditional Probability

The Conditional Probability $P(A|B)$ represents the conditional probability of event A given that event B has already occurred.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Example: For rolling two dice, Suppose we are interested in finding the probability of getting a sum of 8 on the two dice, given that the first die shows an odd number. We'll denote this event as A and B, respectively:
Event A: The sum of the numbers rolled on two dice is 8.
Event B: The first die shows an odd number.
Now, $P(B) = 3/6 = 1/2$
And $P(A \cap B) = 2/36 = 1/18$
Using the formula for conditional probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A|B) = \frac{1/18}{1/2} = 1/9$$

Therefore, the conditional probability of getting a sum of 8 on the two dice, given that the first die shows an odd number, is $1/9$. This means that if we know the first die shows an odd number, the chance of the sum being 8 increases to $1/9$.

Bayes Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Example: Suppose we have dice. One die D1 is fair and the other die D2 is biased. Given that Die D1 is chosen with a probability of $\mathrm{P(D1)} = 0.4$ and Die D2 with $\mathrm{P(D2)} = 0.6$, and we know that:
$P(6|D1)$ = Probability of getting 6 with Die D1 $= 1/6$
$P(6|D2)$ = Probability of getting 6 with Die D2 $= 1/2$
Now, we want to find the probability that the biased die was chosen, given that we got 6, i.e $P(D2|6)$.
Using Bayes' theorem, we can calculate this as follows:

$$P(D2|6) = \frac{P(6|D2) * P(D2)}{P(6)}$$

where $P(6)$ is the probability of rolling 6, regardless of which die is used. This can be calculated as the sum of the possibilities of rolling 6 with Dice A and Dice B:

$$P(6) = P(6|D1) * P(D1) + P(6|D2) * P(D2)$$
$$= (1/6) * (0.4) + (1/2) * (0.6) \approx 0.367$$

Now, we can calculate $P(D2|6)$:

$$P(D2|6) = \frac{P(6|D2) * P(D2)}{P(6)}$$

$$\approx (1/2 * 0.6)/0.367 \approx 0.817$$

Therefore, the probability of a biased die being chosen, given that we rolled 6, is approximately 0.817.

A.4 *Useful Bounds*

Union Bound

For every collection of events $E_1, E_2, ..., E_n$

$$Pr(\bigcup_{i \in [n]} E_i) \leq \sum_{i \in [n]} P(E_i)$$

Markov's Inequality

Let X be a non-negative random variable. For all $t > 0$,

$$P(X \geq t) \leq \frac{E[X]}{t}$$

Chebyshev's Inequality

Let X be a random variable. For all $t > 0$,

$$P[|X - E[X]| \geq t] \leq \frac{Var(X)}{t^2}$$