CSE 426: Cryptography

University of Washington

Andrew Chen

Autumn 2025

Hello and welcome! This is my lecture notes on CSE 426A – Cryptography. This course is the first and only undergrad course on cryptography offered here at UW. The professor is **Stefano Tessaro**, and we meet MWF at **11:30 am** for lectures. The textbook that we are using is **Rafael Pass and Abhi Shelat's A Course in Cryptography**. Also note that theorem names might not necessarily be accurate; it's probably just whatever my textbook / professor said it is.

The goal of these lecture notes is to write **understandable** math. As the great Albert Einstein put it, "If you can't explain it to a six year old, then you don't understand it yourself". The hope is that anyone coming across these notes (like you!) will be able to at least take away the gist of these concepts. Should you find any errors in my mathematics, please contact me at zchen66@uw.edu

Contents

List of Theorems

List of Definitions

1 Lecture 01: Sept. 24th

Today during lecture, we went over basic course preamble and logistics, and was introduced to some background and high-level examples within the field of cryptography.

1.1 Course Preamble

This course will go over the basics of modern day cryptography, with an emphasis on the CS Theory and rigorous mathematics aspect of the field. Students are assessed through 3 ways – problem sets (7), an in-class midterm, as well as an in-class final.

1.2 Cryptography overview

The **goal** of cryptography is to allow systems to achieve their functionalities in the presence of *adversaries* interfering with this goal.

Some basic examples include:

- Secure internet communication
- Wi-Fi security
- Secure messaging
- Storage server
- Exposure notifications (like during covid)
- Cryptocurrencies (which are NOT the same as cryptography!!)
- Multi-party computations

Rigorously speaking, the set of potential strategies that could be employed by adversaries are *infinite*, which makes cryptography "hard". Thus, in this course, we will be focused primarily on **provable cryptography**.

Definition (Provable cryptography)**.** Modern cryptography provided security *guarantees* through mathematical proofs

Remark. This allows us to have science-based claims and well-studied assumptions that can be used to form rigorous **theorems**.

Most importantly, to end off the lecture, we are emphasized that this course focuses on the *principles*, and gives a taste of the practice.

2 Lecture 02: Sept. 26th

Today we covered some basic syntactical things regarding cryptography, and then took a little deeper of a dive regarding the nuance of the concept of "security". The discussion of extracting information and defining what a successful attack looks like changed my perspective a lot. Before, I always thought security was very binary – information was either known or not known, but now it feels much more nuanced.

2.1 Syntax of encryption schemes

Definition (Symmetric cryptography)**.** the **sender** and the **receiver** share a *secret* key and want to communicate securely in presence of an **adversary** who we assume not to know the key

Remark. Many times, the sender is referred to as *Alice*, the receiver as *Bob*, and the adversary *Eve*.

In the class, we will be analyzing cryptographic **schemes**, which is a collection of **algorithms** meant to jointly achieve a particular cryptographic task.

We will specify algorithms using pseudocode and/or diagrams. Algorithms will often be *randomized*, which will be denoted with $\leftarrow\!\!\$$

Example 2.1. To sample a random 128-bit binary string, we write

$$r \leftarrow\!\!\$ \{0,1\}^{128}$$

Definition (Symmetric encryption scheme)**.** a **symmetric encryption scheme** is a triple of algorithms $\Pi = (\mathbf{Kg}, \mathbf{Enc}, \mathbf{Dec})$, where:

- The (*randomized*) key-generation algorithm **Kg** takes no input and outputs a key K.
- The encryption algorithm **Enc** takes the key k and some plaintext M, and outputs a ciphertext $C \leftarrow \mathbf{Enc}(K, M)$
- The decryption algorithm **Dec** is such that $\mathbf{Dec}(K, \mathbf{Enc}(K, M)) = M$ for every plaintext M and key K output by **Kg**.

Message/plaintext space $\mathcal{M}$ is usually understood as $\{0,1\}^*$ or $\{0,1\}^n$ for some $n \in \mathbb{N}$. This naturally means that the scheme can only encrypt / decrypt texts within $\mathcal{M}$.

Example 2.2. Given the following triple of algorithms, determine if it's a *symmetric encryption scheme*.

1: **Procedure** Kg():	1: **Procedure** Enc(K, M):	1: **Procedure** Dec(K, C):
2: **return** 0^{128}	2: **return** M	2: **return** C

Yes, this is a symmetric encryption scheme, because for all plaintext M, we have $\mathbf{Dec}(0^{128}, \mathbf{Enc}(0^{128}, M)) = M$. Personally wouldn't use this though, it's not the most secure...

Bottom line, "syntax / correctness" is defined separately from "security", as security is something that's much *harder* to define.

2.2 The concept of security

We first started with an example of a mono-alphabetic substitution.

Example 2.3. Let us assume $\mathcal{M} = \{A, ..., Z\}^*$, with the Key Generation function being a random one-to-one mapping

$$\pi = \{A, B, ..., Z\} \to \{A, B, ..., Z\}$$

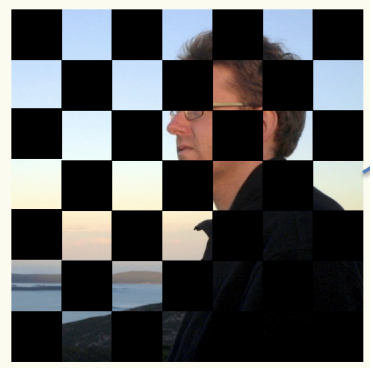
plain	A	B	C	D	...
cipher	N	O	P	Q	...

How many possible keys? Is it secure? Well, obviously $26 \times 25 \times 24 \cdots = 26! = 4.03 \times 10^{26}$ possible keys. Security, however, is always an ill-defined term. We need to specify a bit more.

Definition (Kerckhoffs's principle)**.** An encryption scheme must be secure even if *everything* about it, *except the key*, is public knowledge.

What this means is that even the algorithms themselves should be public knowledge. No security by obscurity. So what exactly is "useful" information? Obviously, the entire plaintext would be really great lol, but sometimes, recovering **partial information** is also usually useful, although usually very **context-dependent**.

For example, take a look at **50%** of a picture of our professor, and notice that you can extract a LOT more than "half" of the information from it.



Remark. An **attack** is successful as long as it recovers *some* useful information about the plaintext(s).

Some common attack settings include

1. **Ciphertext-only attack**, the adversary only sees ciphertexts $C_1^*, C_2^*, ...$, and wants to recover any "useful information" about the plaintexts $M_1^*, M_2^*, ...$
2. **Known-plaintext attack**: Attacker learns also pairs of plaintext and ciphertext $(M_1, C_1), (M_2, C_2), ...$
3. **Chosed-plaintext attack**: Attacker can choose the plaintexts $M_1, M_2, ...$

Remark. A **secure encryption scheme** should *hide all possible useful information* about the plaintext(s).

Remark. Break $\neq$ finding the key!!

We can see this by revisiting our mono-alphabetic substitution from **Example 2.3**.

Since not all letters are equally as likely to be used in the English language (for example, vowels like 'a' 'e' and 'i' are used much more than consonants like 'q'), mono-alphabetic substitution can be broken pretty easily via techniques like *frequency analysis*, especially if the ciphertext is sufficiently long.

There are also many other subtle ways to break a scheme like such. Doesn't feel too "secure" now, does it?

3 Lecture 03: Sept. 29th

Today in lecture we started experimenting with much more of the mathematical rigor behind cryptography. I struggled sometimes with understanding the formalized definitions of security through mathematics. Something new I learned was what a "cryptographic" proof looked like.

3.1 Types of attacks (review)

We started with a discussion on the common ways of attack as discussed towards the end of the previous lecture. To do this, we did an example demonstrating what a plaintext-ciphertext attack could look like.

3.2 Perfect secrecy

We followed up with a new encryption scheme.

Example 3.1. The One-time pad scheme is defined as the following triple of algorithm with $\mathcal{M} = \{0,1\}^n$, $\mathcal{C} = \{0,1\}^n$, $\mathcal{K} = \{0,1\}^n$:

1: **Procedure** Kg():
2: $K \leftarrow\$ \{0,1\}^n$
3: **return** K

1: **procedure** Enc(K, M)
2: **return** $C \leftarrow M \oplus K$

1: **Procedure** Dec(K, C):
2: **return** $M \leftarrow C \oplus K$

Now, is this scheme secure? Recall that our goal is to define security as property that the adversary does not learn anything potentially useful from a ciphertext.

Definition (Shannon's requirement of security)**.** Given some *a-priori* information about the plaintext, the adversary cannot learn any additional information about the plaintext by observing the ciphertext

But wait... detourrrrrrrrr to explore some probability notation:

- $V \leftarrow\$ D$: Sampling V from distribution D
- $V \leftarrow\$ \mathcal{S}$: Sampling V from a set $\mathcal{S}$ uniformly and randomly
- $\Pr_{\text{Experiment}}[\text{Event } A]$: The probability of Event A in a random experiment

Back to security!

Definition (Shannon secrecy)**.** A symmetric encryption scheme $\Pi = (\mathbf{Kg}, \mathbf{Enc}, \mathbf{Dec})$ satisfies **Shannon secrecy** w.r.t. distribution D over the plaintext space $\mathcal{M}$ if for all plaintexts $M* \in \mathcal{M}$ and all ciphertexts $C* \in \mathcal{C}$,

$$\Pr_{K \leftarrow\$ \mathbf{Kg}, M \leftarrow\$ D}[M = M^* | \mathbf{Enc}(K, M) = C^*] = \Pr_{M \leftarrow\$ D}[M = M^*]$$

- $\Pi = (\mathbf{Kg}, \mathbf{Enc}, \mathbf{Dec})$ satisfies **Shannon secrecy** if it satisfies Shannon secrecy w.r.t. *every distribution* D over $\mathcal{M}$.

Definition (Perfect secrecy)**.** A symmetric encryption scheme $\Pi = (\mathbf{Kg}, \mathbf{Enc}, \mathbf{Dec})$ satisfies **perfect secrecy** if for all pairs of plaintexts $M_0, M_1 \in \mathcal{M}$, and all ciphertexts $C \in \mathcal{C}$,

$$\Pr_{K \leftarrow\$ \mathbf{Kg}}[\mathbf{Enc}(k, M_0) = C] = \Pr_{K \leftarrow\$ \mathbf{Kg}}[\mathbf{Enc}(k, M_1) = C]$$

Theorem 3.1. Π *satisfies perfect secrecy if and only if it satisfies Shannon secrecy.*

Proof sketch. Both notions of secrecy imply that the ciphertext distribution is independent from plaintext □

Theorem 3.2. *The one-time pad satisfies perfect secrecy.*

Proof. For all $M, C \in \{0,1\}^n$,

$$\begin{aligned}\Pr_{K \leftarrow\$ \mathbf{Kg}}[\mathbf{Enc}(K, M) = C] &= \Pr_{K \leftarrow\$ \{0,1\}^n}[M \oplus K = C] \\ &= \Pr_{K \leftarrow\$ \{0,1\}^n}[K = C \oplus M] \\ &= \frac{1}{2^n}\end{aligned}$$

□

To provide some *much needed* intuition for the above proof, notice that the result of the equation *isn't dependent on* M at all. What the proof is essentially saying is that regardless of the choice of M, the probability of receiving C as the result of the encryption algorithm will **always be** $1/2^n$.

This matches our definition of **perfect secrecy**, since two plaintexts M_0 and M_1 will have equal probability (precisely $1/2^n$) in encrypting to the same ciphertext C.

So, is the OTP a good encryption scheme now? Well consider the following:

Example 3.2. How to we encrypt $2n$ bits without making the key longer? We can try one potential solution:

$$\mathbf{Enc}(K, M_1 M_2) = \mathbf{Enc}(K, M_1) || \mathbf{Enc}(K, M_2) = M_1 \oplus K || M_2 \oplus K$$

But does this work tho?

Turns out... not really. In fact, it sucks – there are many different flaws. The first of which is that if we xor any two given cipher texts, we end up with the xor of the plaintexts, revealing unwanted information.

$$C_1 \oplus C_2 = (M_1 \oplus K) \oplus (M_2 \oplus K) = M_1 \oplus M_2$$

There's another, more concrete reason as well. Specifically, this scheme no longer satisfies the definition of perfect secrecy. Suppose $M_1 = M_2 = 0^n$, then we have

$$\Pr_{K \leftarrow\$ \{0,1\}^n}[\mathbf{Enc}(K, M_1 M_2) = 0^{2n}] = \frac{1}{2^n}$$

However, suppose $M_1 = 0^n, M_2 = 1^n$, then

$$\Pr_{K \leftarrow\$ \{0,1\}^n}[\mathbf{Enc}(K, M_1 M_2) = 0^{2n}] = 0$$

If we have encrypt both plaintexts using the same key, then there's no way to achieve 0^n when the plaintexts differ, and thus creating different probabilities of encryption **dependant on** the plaintexts themselves. This violates the definition of perfect secrecy.

So actually, unfortunately despite being very expensive, we would *need* to extend the key in this case to maintain Shannon / perfect secrecy.

Let's formalize this thought in the relationship between plaintext and key length necessary for security.

Theorem 3.3. *For every* $\Pi = (\boldsymbol{Kg}, \boldsymbol{Enc}, \boldsymbol{Dec})$ *with perfect secrecy, we have* $|\mathcal{M}| \leq |\mathcal{K}|$.

Proof (by contrapositive). Start by assuming $|\mathcal{M}| > |\mathcal{K}|$.

Now, let $M*$ be an arbitrary plaintext in $\mathcal{M}$.

We define $\mathbf{Enc}(M*)$ to be the set of all possible encryptions of $M*$.

$$\mathbf{Enc}(M*) = \{C \mid \exists K \in \mathcal{K} : \mathbf{Enc}(K, M*) = C\}$$

Now, we pick some arbitrary ciphertext $C* \in \mathbf{Enc}(M*)$.

Let's now define $\mathbf{Dec}(C*)$ to be the set of all possible decryptions of $C*$.

$$\mathbf{Dec}(C*) = \{M \mid \exists K \in \mathcal{K} : \mathbf{Dec}(K, C*) = M\}$$

Using our contrapositive assumption as well as the set we've just created, we must now have $|\mathbf{Dec}(C*)| \leq |\mathcal{K}| < |\mathcal{M}|$. This is because there can be *at most* as many possible decryptions of $C*$ as there are keys in $\mathcal{K}$.

Let's now pick a plaintext $M^{**} \in \mathcal{M} \setminus \mathbf{Dec}(C*)$. Then we have

$$\Pr_{K \leftarrow\$ \mathbf{Kg}}[\mathbf{Enc}(K, M*) = C*] > 0,$$

since we've established that $C* \in \mathbf{Enc}(K, M*)$. However, we also have

$$\Pr_{K \leftarrow\$ \mathbf{Kg}}[\mathbf{Enc}(K, M^{**}) = C*] = 0,$$

since $M^{**} \notin \mathbf{Dec}(C*)$.

Notice now that the probability of the encryption algorithm producing $C*$ varies based on the given plaintext, this violates the definition of perfect secrecy, and is thus a contradiction. □

Remark. From the above theorem and proof, we should note that **perfect secrecy is NOT practical!** In order to achieve perfect secrecy, we would need as much keys as the actual data itself. In other words, to encrypt 1GB of data, we would need 1GB of keys! Super inefficient.

So although perfect security is nice, it's also not at all practical. Instead, we will begin focusing on **Computational Security**, where, despite not being "perfect", schemes are still computationally hard to break, yet they remain efficient and practical to implement.

We ended lecture with a brief summary:

- One-Time Pad (OTP) construction
- Equivalent definitions of Shannon & perfect secrecy

  the key intuition is the independence of ciphertext and plaintext distributions
- OTP achieves perfect seccrecy
- OTP is not secure when encrypting two (or more) messages
- Perfect secrecy requires more keys than plaintexts
- To be practical, we will relax security to be "computational"

4 Lecture 04: Oct. 1st

Today in lecture we went over the basics of block ciphers and what "algorithms" look like in cryptography in the form of oracles. For me, this is the first lecture where concepts are starting to get a bit challenging. The definition of block cipher took me a bit to wrap my head around.

4.1 Block ciphers

First, we started with a recollection of mono-alphabetic substitution. Basically, remember that it sucked. So let's propose some alternatives.

Introducing the **Super-poly-graphic substitution**, where we encrypt a *block* of characters at a time!

AAAAAAAAA	AAAAAAAAB	AAAAAAAAC	...	ZZZZZZZZZ
DIFOKBXEH	HAGKCREDK	HKTOXDEQZ	...	LJIQOVZXI

Notice how this just clears? Because now, statistical analysis is made *so much* more difficult (even known-plaintext attacks are not very useful!)

But here is the worse part – the key lengths become *enormous!* In the above example, we would need $26^7! = 8 \times 10^9!$ keys. For context, that's more than the number of *microseconds* since the big bang... Needless to say, we need something a little more efficient.

Definition (Block cipher (informal))**.** A **block cipher** is a succinct represntation of a substitution table for large blocks

> **Intuition.** *Essentially, rather than having a unique key for each unique block, we have just a limited amount (1 or 2)* "master keys"*, where the master keys will be able to permute all the blocks.*

Definition (Block cipher (formal))**.** A **block cipher** is a function $\mathbf{E}$:$\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ s.t. $\forall K \in \{0,1\}^k$, the mapping $\mathbf{E}_k : X \to \mathbf{E}(K, X)$ is a permutation.

Inverse $\mathbf{E}^{-1} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ of $\mathbf{E}$ s.t. $\mathbf{E}^{-1}(K, Y) = \mathbf{E}_K^{-1}(Y)$.

> **Intuition.** *think like a* giant *look-up table, where every input is mapped to a unique output. that's what the block cipher is. given a block of data, it returns a different block of data of the same length thru some permutation (not of the actual data, but just of the possible outputs)*

Remark. Here, a key will be able to permute each block of plaintext one-to-one and onto (bijectively) to a block of ciphertext. "Permutation" is the same as saying a bijective mapping, which itself must be invertible.

Example 4.1. Is the One-time Pad a block cipher? Well, Let

$$\mathbf{E}:\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$

such that

$$\mathbf{E}(K, X) = K \oplus X$$

And actually, yes! Since

$$\forall K, X \neq X' : \mathbf{E}(K, X) = X \oplus K \neq X' \oplus K = \mathbf{E}(K, X')$$

In other words, for each unique input X, OTP will always produce a unique output X'. This means that the mapping is bijective, and thus is a block cipher.

Example 4.2. We now use block ciphers for substitution. When encrypting the plaintext

$$M = M_1 M_2 M_3 ...$$

for blocks $M_1, M_2, ... \in \{0,1\}^n$, return ciphertext

$$C = C_1 C_2 C_3 ...$$

where $C_i \leftarrow \mathbf{E}(K, M_1)$. Note here that we're using the same key for all blocks!

But why does this work? I thought we've seen that with OTP, using the same key is NOT secure?

For security, we have an **informal security requirement**: The behavior of a secure block cipher is "indistinguishable" from that of a randomly chosen substitution table.

A substitution table from $\{0,1\}^n$ to $\{0,1\}^n$ is a permutation $\pi \in \text{Perm}(\{0,1\}^n)$.

But how exactly do we sample a random permutation? Seems like something a well-written algorithm could do for us!

4.2 Oracles

More convenient to think of the random substitution as an **oracle** denoted $\mathbf{RP}[n]$ which can be queried on input x to learn $\pi(x)$ to a uniformly chosen permutation $\pi : \{0,1\}^n \to \{0,1\}^n$

We should think of **oracles** as an instance of a class in some programming language, where

- **Init()** runs once, at the beginning
- Keeps states (global variables)
- Following that, we can call the **Eval(X)** function with some parameter X to query based on that input.

Let's see an example

Algorithm 1 oracle $\mathsf{RP}[n]$**:**

procedure INIT:
    $S \leftarrow \emptyset$
    for all $X \in \{0,1\}^n$ **do**
        $T[X] \leftarrow\!\!\$\ \{0,1\}^n \setminus S$
        $S \leftarrow S \cup T[X]$

procedure EVAL(X):
    return $T[X]$

Yay pseudocode! But this is quite inefficient at the moment. Notice that it takes 2^n time to sample the table, because we are building the entire table from scratch when we initialize.

We can use a technique called *lazy sampling* to make this much more efficient. The idea is only randomly sample when it's needed (i.e. upon eval). Don't need to do unnecessary work ahead of time!

Algorithm 2 oracle $\mathsf{RP}[n]$**:**

procedure INIT:
    $S \leftarrow \emptyset$
    $T \leftarrow \{\}$

procedure EVAL(X):
    if $T[X] = \perp$ **then**
        $T[X] \leftarrow\!\!\$\ \{0,1\}^n \setminus S$
        $S \leftarrow S \cup T[X]$
    return $T[X]$

Now, back to the task at hand. We want to be able to compare block cipher use with the output of $\mathbf{RP}[n]$. To do this, we introduce another oracles the Keyed Function Oracle.

Algorithm 3 oracle $\mathsf{KF}[\mathbf{E}]$**:**

procedure INIT:
    $K \leftarrow\!\!\$\ \{0,1\}^k$

procedure EVAL(X):
    return $\mathbf{E}[K, X]$

Finally, we ended the lecture with our eyes set on the next goal: For a "good" block cipher $\mathbf{E}$, it should be hard to decide whether we're given access to $\mathsf{RP}[n]$ or $\mathsf{KF}[\mathbf{E}]$,. We'll formalize this next time.

5 Lecture 05: Oct. 3rd

Today we were introduced to our first formal look at an adversarial program. Something new I learned was that distinguishers can be designed with specific malicious intent to work best on certain programs, in order to fully expose their non-random permutative nature.

5.1 Distinguishers

We started with a recollection of the notion of security of block ciphers. Essentially, it should be *hard* to distinguishing the output of $\mathbf{KF}[E]$ and $\mathbf{RP}[n]$

To do this, we introduced a **distinguisher**, our first formal look at a form of an *adversary.*

Definition (Distinguisher)**.** We specify the distinguisher D^O as an algorithm, in *pseudocode*, which can invoke public procedures from an oracle O.

Algorithm 4 Distinguisher

```
1: procedure DISTINGUISHER D^O:
2:     Y_1 ←$ O.Eval(0^n)
3:     Y_2 ←$ O.Eval(1^n)
4:     if Y_1 = Y_2 then
5:         return 1
6:     else
7:         return 0
```

With underlying random experiments, we're always going to have some probability $\Pr[D^O() \Rightarrow 1]$ based on the output of calls to our oracle O.

With this notion of probability, we can now tell how "close" any given two oracles are. For example, if $\Pr[D^{O_1}() \Rightarrow 1]$ and $\Pr[D^{O_2}() \Rightarrow 1]$ are very similar, this would mean that the distinguisher D would have a hard time telling the two apart.

To formalize this thought, we introduce an **advantage**.

Definition (Advantage)**.**

$$\mathrm{Adv}^{\mathrm{dist}}_{O_1,O_2}(D) = |\Pr[D^{O_1}() \Rightarrow 1] - \Pr[D^{O_2}() \Rightarrow 1]|$$

Using an example now,

Definition (PRP Advantage)**.** The **PRP advantage** of D against E is

$$\mathrm{Adv}^{\mathrm{prp}}_{E}(D) = |\Pr[D^{\mathbf{KF}[E]}() \Rightarrow 1] - \Pr[D^{\mathbf{RP}[n]}() \Rightarrow 1]|$$

Here, **PRP** = Pseudo-Random Permutation

From there, we went over an example of a distinguisher with the malicious intent of breaking OTP block ciphers.

Example 5.1. To calculate this distinguisher's advantage on OTP, we first assume $O = \mathbf{RP}[n]$.

Algorithm 5 Distinguisher

```
1: procedure DISTINGUISHER D_2^O:
2:     Y_0 ←$ O.Eval(0^n)
3:     Y_1 ←$ O.Eval(1^n)
4:     if Y_0 ⊕ Y_1 = 1^n then
5:         return 1
6:     else
7:         return 0
```

Doing a little bit of probability calculations, let $Y_0 \xleftarrow{\$} \{0,1\}^n$ and $Y_1 \xleftarrow{\$} \{0,1\}^n \setminus \{Y_0\}$. Since (Y_0, Y_1) is uniformly distributed over all pairs of n-bit strings, we know there must exist $2^n(2^n - 1)$ such pairs to sample from.

Out of the available pairs, there must also exist 2^n of which will produce 1 when xor'd together (all the pairs in which every bit of each string differs from the other).

Therefore, $\Pr[Y_0 \oplus Y_1 = 1^n] = \frac{2^n}{2^n(2^n-1)} = \frac{1}{2^n-1}$, and so we arrive at

$$\Pr[D_2^{\mathbf{RP}[n]} \Rightarrow 1] = \frac{1}{2^n - 1}$$

Now, let's take $O = \mathbf{KF}[\mathbf{E}]$, where $\mathbf{E}[K, X] = K \oplus X$.

Notice that $Y_0 = K$, since any string xor'd with 0^n will always remain unchanged. Additionally, $Y_1 = K \oplus 1^n$.

From here, we see that

$$Y_0 \oplus Y_1 = K \oplus (K \oplus 1^n) = 0^n \oplus 1^n = 1^n,$$

and thus, $\Pr[Y_0 \oplus Y_1 = 1^n] = 1$, so we arrive at

$$\Pr[D_2^{\mathbf{KF}[E]} \Rightarrow 1] = 1$$

Finally, subtracting the two, we get the PRP advantage

$$\mathrm{Adv}_E^{\mathrm{prp}}(D_2) = |\Pr[D^{\mathbf{KF}[E]}() \Rightarrow 1] - \Pr[D^{\mathbf{RP}[n]}() \Rightarrow 1]| = 1 - \frac{1}{2^n - 1}$$

Wow! For such a simple distinguisher, we have quite an atrocious prp advantage. This just means that OTP does not behave like a random permutation, and therefore, is not a good block cipher.

> **Intuition.** *To clarify, as proven before, OTP satisfies perfect secrecy and is thus a fantastic encryption scheme. However, it doesn't make for a good block cipher since, when repeatedly used to create a permutation, the outcome is very predictable (a linear relationship between the input and outputs).*

Ideally, we should have $\text{Adv}_E^{\text{prp}}(D)$ is *very small* (e.g., $\leq 2^{-k}$) for all D. We will soon find that this is quite infeasible.

> **Intuition.** *Otherwise, a secure block cipher would require us to test it with every distinguisher possible, which is very infeasible.*

So the next best thing is to get real subjective. We say that we want $\text{Adv}_E^{\text{prp}}(D)$ to be "very small" for "feasible" D. If this is true, we say that E is a *pseudorandom permutation*. There are two options of security here – concrete vs. asymptotic.

We will mostly be focused on the latter in this class.

5.2 Asymptotic security

We start off with some definitions.

Definition (Concrete security)**.** For $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, we want that

$$\text{Adv}_E^{prp}(D) \leq \epsilon(k)$$

for every D running in time at most $t(k)$.

Definition (Asymptotic security)**.** For $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, we want

$$\text{Adv}_E^{prp}(D) \leq \epsilon(k)$$

for every D running in time *polynomial* in k, where $\epsilon(k)$ is a *negligible function*.

Definition (Negligible function)**.** A function $f : \mathbb{N} \to \mathbb{R}$ is **negligible** if for all $c \in \mathbb{Z}$, there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$, we have

$$f(k) < \frac{1}{k^c}$$

As a note, **asymptotic security** is mostly used in theoretical cryptography, where as **concrete security** is usually used in applied cryptography. Within this course, as stated above, we will mostly be focusing on asymptotic.

6 Lecture 06: Oct. 6th

To day's lecture was a lot of examples and english definitions. We started with walking thru the entirety of AES, then moved from Block ciphers to the notion of *semantic security*. I'm still not super comfortable with this idea at this time because like how is it different from Shannon?

6.1 Block ciphers (cont'd)

We started, as usual, with a recap of the block cipher, distinguishers, advantages, and what it means to be secure (asymptotic security).

We then went over a brief overview of the Advanced Encryption Scheme (AES). In this class, we will assume that **AES = AES-128** is a $\left(t, \frac{t}{2^{126}}\right)$-PRP for any $t \leq 2^{126}$

But are block ciphers good encryption schemes? The issue is that we can only encrypt short n-bit messages. Naively, we can use block ciphers as a *substitution table*, much like the initial envisioning of their use.

This is often referred to as **Electronic Codebook (ECB)** mode, where we take n-bit blocks of the plaintext and substitute them with n-bit ciphertexts which are the output of some block cipher. But we quickly see that there are some security issues with this approach.

Example 6.1. Assume we know Alice is sending one of the two following texts:

M_1 = "HELLO WORLD BYBYE WORLD "
M_2 = "HELLO WORLD HELLO WORLD "

And we see the following ciphertext:

C = ee 04 f4 45 54 da 20 d6 0b b1 a5 96 cc 64 6d 1f 2c 52 6d 22 64 a9 2f 66

We can easily tell that M_1 was the original plaintext, as there must have been some repetition had M_2 been the original plaintext.

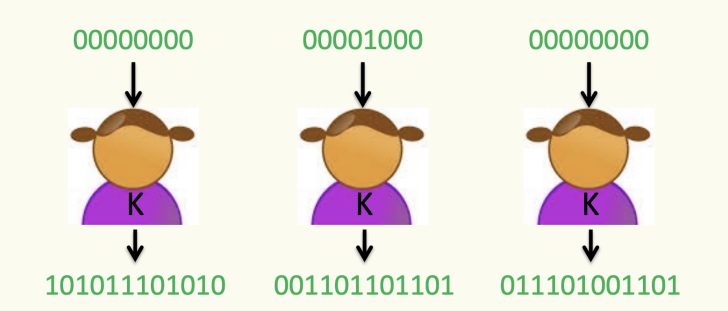
Well, this is certainly not "secure". Let's once again formalize this thought.

6.2 Semantic security

The goal of semantic security is to make sure the ciphertexts leak *nothing* about the plaintexts, other than what we knew to begin with.

And to achieve this, we need to make sure that every time we encrypt something (regardless of new or old messages), the ciphertexts looks random and independent in the eyes of some *computationally bounded* adversary.

> **Intuition.** *This basically extends the idea of Shannon to now encrypting "multiple plaintexts" (or multiple n-bit blocks of the same plaintext).*



And so to achieve this new notion of security, we introduced another encryption technique known as the **One-Block Randomized Counter Mode (1$CTR)**.

Example 6.2. The 1$CTR is defined as the following:

procedure K_G
  $K \xleftarrow{\$} \{0,1\}^n$
  return K

procedure $\text{E}_\text{NC}(K, M)$
  $R \xleftarrow{\$} \{0,1\}^n$
  $C \leftarrow (R, \mathbf{E}(K, R) \oplus M)$
  return C

procedure $\text{D}_\text{EC}(K, C)$
  $(R, C') \leftarrow C$
  return $C' \oplus \mathbf{E}(K, R)$

Couple things to notice: (1) The ciphertext now has length $2n$, since it must also start with the randomly sampled R, and (2) The same plaintext will be encrypted to two different ciphertexts

7 Lecture 07: Oct. 8th

Today we went over a formalized variant of semantic security known as the IND-CPA security, which allowed us to formalize many of the notions mentioned in the previous lecture. During the actual lecture, everything felt all over the place, but upon reviewing, I was able to piece together a lot of the intuition behind ind-cpa and why we use certain techniques.

7.1 IND-CPA Security

To start formalizing semantic security, we will start with a variant known as **IND-CPA security**, which stands for

Indistinguishability under a **C**hosen **P**laintext **A**ttack

And so to further understand this, let's take a trip down memory lane and recall the definition of *perfect secrecy*:

$$\Pr_{K \leftarrow \$ \mathbf{Kg}}[\mathbf{Enc}(K, M_0) = C] = \Pr_{K \leftarrow \$ \mathbf{Kg}}[\mathbf{Enc}(K, M_1) = C]$$

When we first introduced this concept, we mentioned that producing the *exact same distribution* is quite infeasible in reality. But also at the same time, this notion of "perfect" only applies when we're encrypting a single text (and not multiple plaintexts, as we want to do now).

> **Intuition.** *Essentially, this definition is both very strong (in rigor) but also very weak (in practicality). As we'll see in a second, we'll push and pull on this a little bit to fit our case better.*

And so to extend this idea of "perfection" to semantic security, we want to allow for:

- *Multiple* encryptions
- *Adaptive* choice of messages
- *Relaxing strict equality* of distributions

> **Intuition.** *If you give me two messages M_0 and M_1 (chosen plaintext attack), and I encrypt one of them for you, you cannot tell which one!*

Definition (IND-CPA Security)**.** To truly formalize this, we define the following oracles:

Algorithm 6 Left World

```
oracle LR_0[Π]:

procedure INIT:
    K ←$ Kg()

procedure ENCRYPT(M^0, M^1):
    C ←$ Enc(K, M^0)
    return C
```

Algorithm 7 Right World

```
oracle LR_1[Π]:

procedure INIT:
    K ←$ Kg()

procedure ENCRYPT(M^0, M^1):
    C ←$ Enc(K, M^1)
    return C
```

Intuition. *The lore behind the names of these oracles is that we're "sending" some distinguisher into one of two worlds (left or right), and ideally they will not be able to distinguish which world they've been "sent to" (it's one big analogy)*

The **ind-cpa advantage** of some distinguisher D against Π is

$$\mathrm{Adv}^{\text{ind-cpa}}_{\Pi}(D) = |\Pr[D^{LR_0[\Pi]} \Rightarrow 1] - \Pr[D^{LR_1[\Pi]} \Rightarrow 1]|$$

Π is (t, ϵ)**-IND-CPA secure** if $\mathrm{Adv}^{\text{ind-cpa}}_{\Pi}(D) \leq \epsilon$ for every D running in time at most t

Π is **IND-CPA secure** if $\mathrm{Adv}^{\text{ind-cpa}}_{\Pi}(D)$ is negligible for every polynomial-time D.

In fact, we have the following theorem

Theorem 7.1. *If* Π *is deterministic, then it is not IND-CPA secure.*

Needless to say, IND-CPA security is a *massively* powerful property that implies many many security requirements. In fact, it implies all of the following

- It is infeasible to recover the key
- It is infeasible to recover the plaintext

- It is infeasible to detect whether plaintexts satisfy a certain given property
- ...

Great we get it blah blah blah it's a really powerful security requirement. But how do we achieve it? Well, recall from the previous lecture the definition of the One-Block Randomized Counter Mode, 1$CTR:

1: **procedure** KG
2: $K \leftarrow\$ \{0,1\}^n$
3: **return** K

1: **procedure** ENC(K, M)
2: $R \leftarrow\$ \{0,1\}^n$
3: $C \leftarrow (R, \mathbf{E}(K, R) \oplus M)$
4: **return** C

1: **procedure** DEC(K, C)
2: $(R, C') \leftarrow C$
3: **return** $C' \oplus \mathbf{E}(K, R)$

To provide some much needed intuition, let's take a closer look at the encryption algorithm.

Algorithm 8 1$CTR Encryption

1: **procedure** ENC(K, M)
2: $R \leftarrow\$ \{0,1\}^n$
3: $C \leftarrow (R, \mathbf{E}(K, R) \oplus M)$
4: **return** C

Couple things to notice here to provide some intuition as to why this satisfies IND-CPA security:

1. Every time we encrypt a new message, we are going to pick a new R with *very high* probability
2. Since we're picking a new R (almost) every time, if $\mathbf{E}$ is some PRP, then $\mathbf{E}(K, R)$ will "look random", thus making the actual encryption step (line 3) a "new" one-time pad with every encryption. Very efficient at hiding M.
3. Note that if we were to accidentally re-use a value of R, then we would be reusing $\mathbf{E}(K, R)$, and the same issues with re-using a one-time pad would occur again. *But this is very unlikely!*

We then followed up with a massive theorem that we will prove in the near future.

Theorem 7.2. *If* $\boldsymbol{E}$ *is a* (t, ϵ)*-PRP, then* ***1$CTR*** *is* (t, δ)*-IND-CPA secure, where*

$$\delta = 2\epsilon + t^2/2^n$$

One other thing to notice about the *decryption* algorithm for 1$CTR is the fact that we never end up having to invert our block cipher $\mathbf{E}$. In fact, the notion that block ciphers *must be invertible* is an outdated thought that used to be enforced back when everything was in ECB mode, and thus needed invertibility in order to be decrypted.

Since that's not the case anymore, we would ideally want block cipher outputs to be **uniform and truly independent**, and that's not really the case with any sort of permutations. So let's ditch the PRP idea, and move onto an abstraction that's a little more useful.

Algorithm 9 Random Function Oracle

```
Oracle RF[m, n]:

procedure Init:
    T ← {}

procedure Eval(X):
    if T[X] =⊥ then
        T[X] ←$ {0,1}^n
    return T[X]
```

Notice that our random function is uniform and truly independent, and thus is probably a better abstraction to compare the output of our block cipher with. Let's formalize this thought.

Definition (PRF Advantage)**.** The **PRF Advantange** of D against $\mathbf{E}$ is

$$\mathrm{Adv}^{\mathrm{prf}}_{E}(D) = |\Pr[D^{\mathbf{KF}[E]}() \Rightarrow 1] - \Pr[D^{\mathbf{RF}[m,n]}() \Rightarrow 1]|$$

Remark. If n is "large enough" the notion of PRP and PRF are essentially equivalent. Here, "large enough" is defined as $n = \omega(\log k)$

$\mathbf{E}$ is a (t, ϵ)-PRF if $\mathsf{Adv}^{\mathrm{prf}}_{\mathbf{E}}(D) \leq \epsilon$ for every D running it time at most t.

$\mathbf{E}$ is a PRF if $\mathsf{Adv}^{\mathrm{prf}}_{\mathbf{E}}(D)$ is negligible (in k) for every D running in polynomial time (in k)

We ended lecture on a pretty important theorem.

Theorem 7.3 (Switching lemma). *Let $\boldsymbol{E} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher, then for every distinguisher D making q queries,*

$$\left|\mathsf{Adv}^{prf}_{\boldsymbol{E}}(D) - \mathsf{Adv}^{prp}_{\boldsymbol{E}}(D)\right| \leq \frac{q^2}{2^{n+1}}$$

8 Lecture 08: Oct. 10th

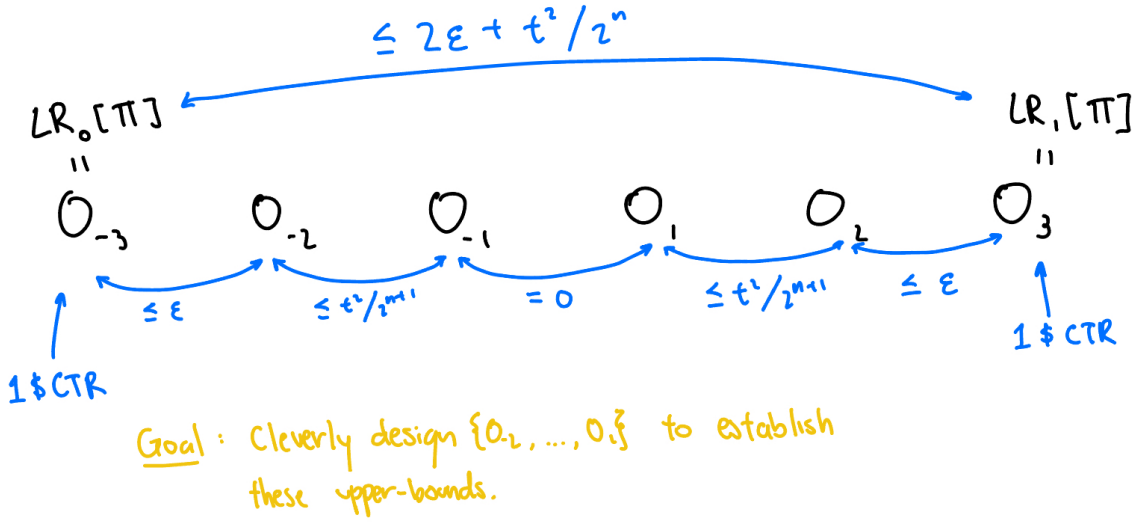
Today in lecture we went over our first formal notion of a security proof. Surprising, I actually was following along for most of it. I definitely need to brush up on my definitions like ind-cpa and things, but overall it was actually pretty understandable.

8.1 1$CTR Security

We began with a quick review on IND-CPA security and the 1$CTR encryption. The goal of the lecture will be to formally and rigorously prove that 1$CTR is IND-CPA Secure.

Proof. To begin, the overarching idea of the proof is to show that the advantage of two oracles, $O_{-3} = LR_0[\Pi]$ and $O_3 = LR_1[\Pi]$ as defined by 1$CTR, is no greater than $2\epsilon + t^2/2^n$ and thus negligible. (see **theorem 7.2** for more details on definition)

To accomplish this, we are going to split the two oracles up into many mini-incremental oracles $O_{-2}, ..., O_2$, and show that the sum of the advantages between all pairs of two adjacent incremental oracles add up to no greater than $2\epsilon + t^2/2^n$.



Let's now formally introduce our contestants!

Algorithm 10 Oracle O_{-3} aka $LR_0[\Pi]$

Oracle O_{-3}:

procedure INIT:
    $K \leftarrow\$ \mathbf{Kg}()$

procedure ENCRYPT(M^0, M^1):
    $R \leftarrow\$ \{0,1\}^n$
    $C \leftarrow (R, \mathbf{E}(K, R) \oplus M^0)$
    return C

Algorithm 11 Oracle O_{-2}

Oracle O_{-2}:

procedure INIT:
    $T \leftarrow \{\}$

procedure ENCRYPT(M_0, M_1):
    $R \leftarrow\$ \{0,1\}^n$
    if $T[R] = \perp$ **then**
        $T[R] \leftarrow\$ \{0,1\}^n$
    $C \leftarrow (R, T[R] \oplus M^0)$
    return C

> **Intuition.** *Notice how O_{-2} looks very similar to the **RF** oracle? This makes it so that distinguishing between O_{-3} and O_{-2} is not easier than distinguishing between the output of **E** and some random function, which we can do with advantage $\leq \epsilon$*

More formally, we can rewrite both oracles to be

Oracle O_{-3}:

procedure INIT:

procedure ENCRYPT(M^0, M^1):
    $R \leftarrow\$ \{0,1\}^n$
    $C \leftarrow (R, \mathbf{KF}[E].\texttt{Eval}(R) \oplus M^0)$
    return C

Oracle O_{-2}:

procedure INIT:

procedure ENCRYPT(M_0, M_1):
    $R \leftarrow\$ \{0,1\}^n$
    $C \leftarrow (R, \mathbf{RF}[n,n].\texttt{Eval}(R) \oplus M^0)$
    return C

From here, notice that both oracles look the exact same aside from the "sub-oracle" they call upon. Thus, we can represent O_{-3} and O_{-2} as a single oracle O^F with $F = \mathbf{KF}[E]$ or $F = \mathbf{RF}[n,n]$

Algorithm 12 O^F

Oracle O^F:

procedure INIT:

procedure ENCRYPT(M_0, M_1):
    $R \leftarrow\$ \{0,1\}^n$
    $C \leftarrow (R, \mathbf{F}.\texttt{Eval}(R) \oplus M^0)$
    return C

Now, we can apply a tricky perspective change. Before we recognized that these oracles can be factored to be the same, we would have needed to run some distinguisher D on both oracles to determine the advantage between the two.

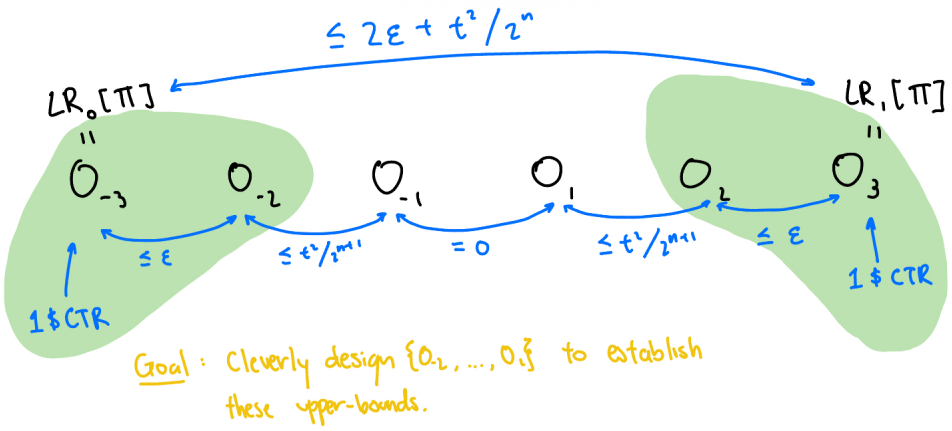
However, now that we combined both oracles into one with the only difference being the parameter function, we can simply treat the entire distinguisher *and* oracle as one! In other words,

$$\Pr[D^{O^{\mathbf{KF}[E]}} \Rightarrow 1] = \Pr[(D^O)^{\mathbf{KF}[E]} \Rightarrow 1]$$
$$\Pr[D^{O^{\mathbf{RF}[n,n]}} \Rightarrow 1] = \Pr[(D^O)^{\mathbf{RF}[n,n]} \Rightarrow 1]$$

where D^O is the "new" distinguisher. And with this in mind, we can see that the advantage between O_{-3} and O_{-2} is actually no different than the advantage of a prf on some block cipher $\mathbf{E}$, just as we had predicted with our intuition!

$$\mathsf{Adv}^{\text{dist}}_{O_{-3},O_{-2}}(D) = \mathsf{Adv}^{\text{dist}}_{\mathbf{KF}[E],\mathbf{RF}[n,n]}(D^O) = \mathsf{Adv}^{\text{prf}}_{\mathbf{E}}(D^O)$$

Since we're given that $\mathbf{E}$ is a (t,ϵ)-PRF, if D^O runs in time t when D runs in time t, then we have $\mathsf{Adv}^{\text{prf}}_{\mathbf{E}}(D^O) \leq \epsilon$, and we're done with the first part of our proof!



Let's now introduce our final contestant.

Algorithm 13 Oracle O_{-1}

Oracle O_{-1}:

procedure INIT:
    $T \leftarrow \{\}$

procedure ENCRYPT(M_0, M_1):
    $R \leftarrow\$ \{0,1\}^n$
    $T[R] \leftarrow\$ \{0,1\}^n$
    $C \leftarrow (R, T[R] \oplus M^0)$
    return C

Before we dive into the final calculation, let's quickly notice that we should have

$$\mathsf{Adv}^{\text{dist}}_{O_{-1},O_1}(D) = 0$$

Why? Because every encryption is a *fresh* one-time pad, and we know one-time pads are perfectly secret!

Moving on, let's once again try to make O_{-2} and O_{-1} look as similar as possible. More formally, we can rewrite both oracles to be

Oracle O_{-2}:

procedure INIT:
    $T \leftarrow \{\}$

procedure ENCRYPT(M_0, M_1):
    $R \leftarrow\$ \{0,1\}^n$
    if $T[R] = \perp$ **then**
        $T[R] \leftarrow\$ \{0,1\}^n$
    else
        bad $\leftarrow 1$
    $C \leftarrow (R, T[R] \oplus M^0)$
    return C

Oracle O_{-1}:

procedure INIT:
    $T \leftarrow \{\}$

procedure ENCRYPT(M_0, M_1):
    $R \leftarrow\$ \{0,1\}^n$
    if $T[R] = \perp$ **then**
        $T[R] \leftarrow\$ \{0,1\}^n$
    else
        bad $\leftarrow 1$
        $T[R] \leftarrow\$ \{0,1\}^n$
    $C \leftarrow (R, T[R] \oplus M^0)$
    return C

Notice that the only difference between the two oracles now is that in O_{-1}, we set $T[R] \leftarrow \{0,1\}^n$ even after "bad" had been assigned 1. We say that O_{-2} and O_{-1} are "equivalent-until-bad".

We can now see that

$$\mathsf{Adv}^{\text{dist}}_{O_{-2},O_{-1}}(D) \leq \Pr[D^{O_{-2}} \text{ sets bad} = 1] = \Pr[D^{O_{-1}} \text{ sets bad} = 1]$$

Intuition. *Given some distinguisher D, the only way that D can gain advantage over O_{-2} and O_{-1} is when they start to behave differently (otherwise indistinguishable), and since they behave differently with probability $\Pr[D^{O_{-2}}$ sets bad $= 1]$, we know that the advantage must be bounded by that same probability.*

Since D makes at most t encrypt queries, and since the values of R are generated uniformly and bad $\leftarrow 1$ is set only when the two oracles collide, we have from the previous lecture that

$$\begin{aligned}\mathsf{Adv}^{\text{dist}}_{O_{-2},O_{-1}}(D) &\leq \Pr[D^{O_{-2}} \text{ sets bad} = 1] \\ &\leq p_{\text{coll}}(t, \{0,1\}^n) \leq \frac{t^2}{2^{n+1}}\end{aligned}$$

This actually gives us enough information to conclude our *entire* proof! Applying triangle inequality to the definition of advantage, we see that

$$\begin{aligned}\mathsf{Adv}^{\text{ind-cpa}}_{\Pi}(D) &= \mathsf{Adv}^{\text{dist}}_{O_{-3},O_3}(D) \\ &\leq \mathsf{Adv}^{\text{dist}}_{O_{-3},O_{-2}}(D) + \mathsf{Adv}^{\text{dist}}_{O_{-2},O_{-1}}(D) + \mathsf{Adv}^{\text{dist}}_{O_{-1},O_1}(D) \\ &\quad + \mathsf{Adv}^{\text{dist}}_{O_1,O_2}(D) + \mathsf{Adv}^{\text{dist}}_{O_2,O_3}(D) \\ &\leq \epsilon + \frac{t^2}{2^{n+1}} + 0 + \frac{t^2}{2^{n+1}} + \epsilon \\ &\leq 2\epsilon + \frac{t^2}{2^n}\end{aligned}$$

which proves the desired claim. □

Yeah, this entire lecture was dedicated to this one massive ahh proof. Goodbye.

9 Lecture 09: Oct. 13th

Today in lecture we covered the idea of when a plaintext isn't exactly n-bits, moving the discussion to a more "real-world" approach of cryptography. And to discuss this, we generalized CTR and even introduced new techniques and a new encryption scheme. I liked this lecture.

9.1 1$CTR Security (cont'd)

We started lecture today with a review, once again, of pseudorandom function security, negligible functions, and what it means to be asymptotic vs concrete.

We then did a quick review of the theorem we spent the entire previous lecture proving. In fact, there is now a corollary

Corollary 9.0.1. *If* $\boldsymbol{E}$ *is a PRF, and* $n = \omega(\log k)$*, then* ***1$CTR*** *is IND-CPA secure.*

What we're saying, and what we need to prove here is that CTR needs to be negligible for every polynomial time distinguisher. Let's show that.

Proof. We begin by fixing any polynomial $t(k)$. Remember that everything should be a function of the key length k.

This means now there exists a negligible function $\epsilon(k)$ where $\mathbf{E}$ is a $(t(k), \epsilon(k))$-PRF. And from the theorem we proved, we know also that 1$CTR is $(t(k), \delta(k))$-IND-CPA secure for some $\delta(k) = 2\epsilon(k) + t(k)^2/2^n$.

Recall from the homework that the sum of any two negligible function is also negligible, meaning $2\epsilon(k)$ is negligible.

From here, notice that if $n = \omega(\log k)$, then it implies that once we apply the definition, we have

$$2^n > k^c$$
$$\log n > c \log k$$

showing that $1/2^n$ is negligible as well. $t(k)^2$ is still a polynomial, and thus multiplying it by a negligible $1/2^n$ still remains negligible.

Finally, δ is the sum of two negligible functions, $2\epsilon(k)$ and $t(k)^2/2^n$, meaning it must also be negligible. This concludes our proof. □

> **Intuition.** *the intuition behind why we do these "generalized" statements is to make the language concrete and abstract out the nitty-gritty details of all that math and notation. This is the key point in the difference between concrete security (the theorem) and asymptotic security (the corollary).*

9.2 Arbitrarily long messages

At the end of the day, 1$CTR is still a *one*-block cipher, meaning we can only encrypt messages of up to n-bits based on the block cipher that's being used. How can we extend this to be able to encrypt arbitrarily long plaintexts (i.e., $\mathcal{M} = \{0,1\}^*$)?

Let's take a look back at our left-right oracles for IND-CPA. What happens when $|M^0| \neq |M^1|$? Well, because we're now working in this arbitrarily long plaintext space, our ciphertext lengths will actually always be different dependant on the size of the input (this is inevitable! pigeon hole).

And because of this, we need to loosen our definition of IND-CPA a bit and say that, when working in an arbitrary plaintext space, we *allow* the ciphertext to leak the length of the plaintext!

Let's update our oracle definition to reflect this.

Algorithm 14 Left-Right World

```
1: oracle LR_b[Π]:        //b ∈ {0,1}
2:
3: procedure INIT:
4:     K ←$ Kg()
5:
6: procedure ENCRYPT(M^0, M^1):
7:     if |M^0| ≠ |M^1| then
8:         return ⊥
9:     C ←$ Enc(K, M^b)
10:    return C
```

Given that we now need to work in the arbitrary plaintext space, like mentioned before, we would also need to generalize Counter-Mode Encryption (CTR) to work for multiple blocks. We begin with some notation.

Notation. For String $X \in \{0,1\}^n$ and natural number $a \in \mathbb{N}$, define $X + a$ as the n-bit string obtained by:

1. Intrepreting X as the binary encoding of some integer b_X
2. Compute the binary encoding of $Y = b_X + a$
3. Let $X + a$ be the n least significant bits of Y (i.e., mod value by 2^n)

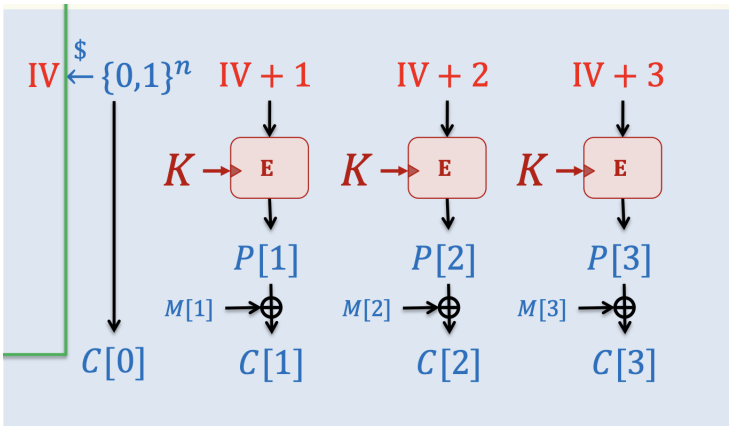
Fantastic! Let's now generalize CTR.

Algorithm 15 Generalized Counter-Mode Encryption

procedure Enc(K, M):
  Split M into blocks $M[1], ..., M[l]$
  // all blocks (except possibly $M[l]$) are n-bits

  $IV \xleftarrow{\$} \{0,1\}^n$
  $C[0] \leftarrow IV$
  for $i = 1$ to l **do**
    $P[i] \leftarrow \mathbf{E}(K, IV + i)$
    $C[i] \leftarrow M[i] \oplus P[i]$
  return $C = C[0]C[1]...C[l]$

Intuition. *Here's a diagram!*



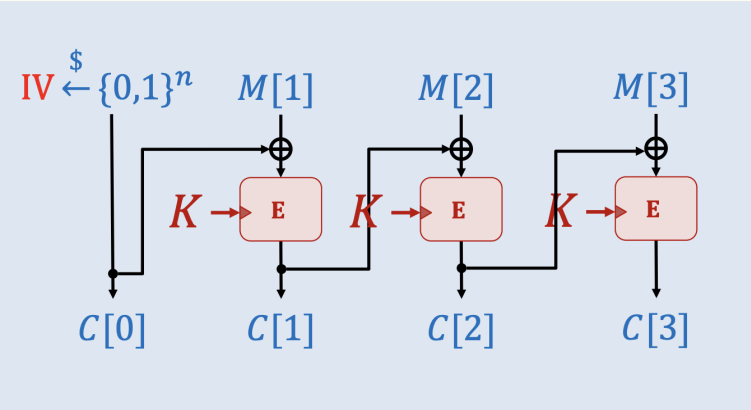
While we're at it, we also introduced a *different* type of encryption scheme known as **Cipher Block Chaining (CBC)**. It's pretty bad for legacy reasons, but it's definitely interesting.

Algorithm 16 Generalized Cipber Block Chaining

procedure Enc(K, M):
  Split M into blocks $M[1], ..., M[l]$
  // all blocks *must be* n-bits

  $IV \xleftarrow{\$} \{0,1\}^n$
  $C[0] \leftarrow IV$
  for $i = 1$ to l **do**
    $C[i] \leftarrow \mathbf{E}(K, M[i] \oplus C[i-1])$
  return $C = C[0]C[1]...C[l]$

Intuition. *Here's another diagram!*



There's a couple things that CBC does which differs with CTR. The most important of which is the concept of "padding". Here, if M isn't exactly a multiple of n-bits, then we need a way to pad it up to a multiple.

There are many ways of doing this, but we need to be careful (for example, something like padding with 0's won't work). We introduced a concept of **PKCS 7 Padding**:

1. Look at how many bytes $k \in \{1, 2, ..., 16\}$ are missing

   `65 6E 74 73 21 00`, $k = 10 =$ `0x0A`

2. Fill the remaining k bytes with the byte value of k

   `65 6E 74 73 21 00 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A`

To decode, we recover the value of the last byte k, and remove the last k bytes given that they're all equal.

One glaring issue we could ask is that "what if the last 10 bytes of the original plaintext just happens to be all `0A`?" To resolve this issue, we would simply append another block of 16 bytes of `0x10` to all data that's already a multiple of 16 to begin with. (yeah this is stupid but wtv it works).

To end off lecture, prof emphasized that despite both CBC and CTR are IND-CPA secure, he spent like 2 minutes shitting on CBC lol. The main point is

- CBC offers roughly the same security as CTR
- For historical reasons, CBC is more popular that CTR
- CTR is potentially faster (can be parallelized)
- CBC requires padding, while CTR does not

So why do we learn this? Because CBC will help us identify a flaw in the definition of IND-CPA security, and as we'll see in the next lecture, under some circumstances, IND-CPA is not "all-powerful".

10 Lecture 10: Oct. 15th

Today in lecture we learned a new mode of attack and some new definitions. It was like a start of a new arc in anime with motivating examples and such. For me this was an incredibly conceptually challenging lecture for me and I found it pretty hard to follow along to the logic of a padded oracle attack, and the INT-CTXT definitions also threw me off. I hope to see more concrete examples in the next lecture to build more intuition.

10.1 Padding Oracle Attacks

We started lecture today with a review of IND-CPA Security, with the addition about revealing plaintext length as discussed on Monday. Then, as promised, we discussed the limits of IND-CPA Security. Mainly,

- IND-CPA security assumes that the adversary only *observes* the encrypted data
  - We call such adversaries **passive**
- Real world attackers can be **active**
  - Rather than influencing the sender to do something (passive), the adversary can also modify the things being sent to the receiver (active)
- An example of an active attack is the **padding-oracle attacks**

Example 10.1. We introduced the concept with a motivating example: CAPTCHAs. They're used to ensure the visitors of some web-app is human. The workflow is as follows:

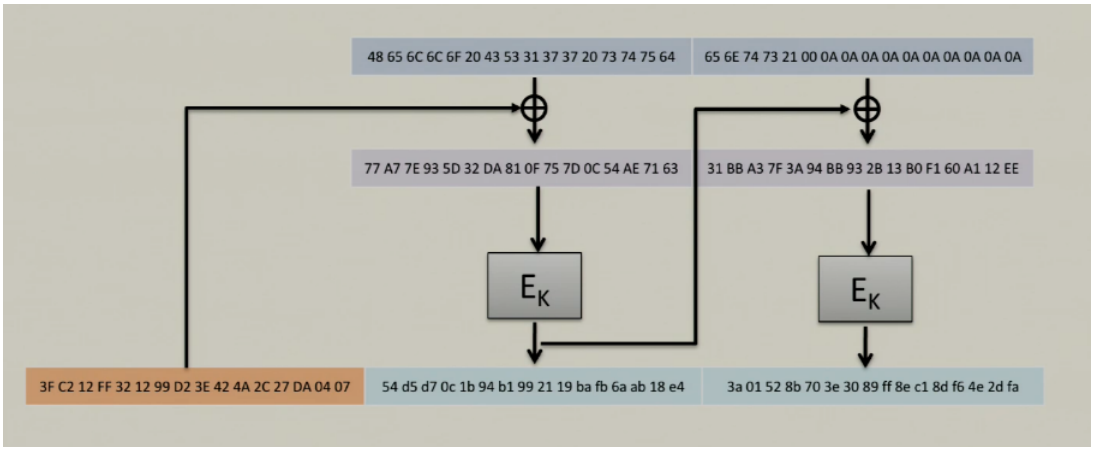
1. The server generates some captcha text image and distorts it
2. To be able to verify the answer later, the website encrypts the answer with a key to get some ciphertext C.
   - The server does this to be "stateless", so that it doesn't have to manually store every single user's captcha solutions
3. The server sends both the image and C to the user (albeit the user cannot see C).
4. The user sends back their guess and C, at which time the server will decrypt C and check it against the user's guess

The security concern occurs on step 4, as a malicious user, rather than guessing the captcha, can instead send back arbitrary ciphertexts $C' \neq C$ to try to learn more about the underlying encryption.

So what exactly is this process in trying to "learn more"? That's where a **padding oracle** comes in. Before I start describing the attack setting, just making a note that we will be assuming the encryption scheme to be CBC. The goal of the attack is to show certain

weaknesses of IND-CPA, and CBC (being a ind-cpa secure encryption scheme) is a perfect candidate for this.

Say we had some $K \leftarrow \mathbf{Kg}()$ and some $C^* \leftarrow \texttt{cbc-enc}(K, M^*)$. Eve's goal is to recover M^*. They will achieve this by submitting different ciphertexts C_i to the padding oracle, at which point the oracle will return with whether or not decryption using C_i will lead to a padding error. Take a look:



Because of the way CBC is designed, if we were to change the *last byte of the first ciphertext block* (call this byte Y), the affected plaintext byte (post decryption) (call this byte X) will be the last byte of the plaintext.

To produce a *valid padding* in this case, we must have $Y = \texttt{0x0A}$ or $\texttt{0x01}$. The adversary knows this and must figure out a way to use this to their advantage.

Take a good look at the example image again, what if we replaced Y instead with $\texttt{E4} \oplus \texttt{0A} \oplus \texttt{01} = \texttt{EF}$? Let's call the output $Z = X \oplus Y$. We know that we now must have

$$Z \oplus (\texttt{E4} \oplus \texttt{0A} \oplus \texttt{01}) = (Z \oplus \texttt{E4}) \oplus (\texttt{0A} \oplus \texttt{01}) = \texttt{0A} \oplus \texttt{0A} \oplus \texttt{01} = \texttt{01}$$

We've produced a valid padding. And, the key thing to notice here is, we've done it with a dependence on the **original plaintext byte** `0A`!

This means any adversary can now do the same. Let's formalize this procedure.
To recover the last byte of the last block: for all possible guesses X:

1. Change the last byte Y of the second-last ciphertext block to $X \oplus Y \oplus \texttt{01}$
2. Submit resulting ciphertext to padding oracle
    - Now, if the padding oracle says ciphertext has valid padding, take X as the value of that last byte)

The key takeaway here is NOT to motivate us to find better padding, or to "just use CTR". The key takeaway here is that this reveals a security flaw in an IND-CPA secure cipher! And

since such an attakc exists, that means *none* of the other IND-CPA ciphers are completely secure in the same way.

We're in need of a new security goal, it seems. We should require that "it must be *infeasible* to produce a new ciphertext which decrypts to a valid plaintext, even after seeing a large number of valid ciphertexts for the same key."

Ciphertext integrity, INT-CTXT security, should really guarantee that

1. Adversary cannot inject encrypted trafic that was not originally sent by Alice
2. Adversary also cannot learn anything useful by submitting shitty ciphertexts to Bob
    - Defeats padding-oracle attacks
    - Referred to as "Chosen-ciphertext security"

11 Lecture 11: Oct. 17th

Today in lecture we formally introduced the idea of data integrity, which is another aspect of cryptography that's different from the goal of simply protecting the confidentiality of data. To talk more about this, we discussed hash functions and collision resistance.

11.1 Data Integrity and Hash Functions

We began lecture with some motivation about what is to come next. Essentially, thus far, we've studied schemes with the sole purpose of protecting the *confidentiality* of the data- to make sure people can't crack our "secret codes".

However, cryptography also plays a very important role in protecting the *integrity* of the data as well- to make sure our messages can't be tampered with.

We began with a motivating example story.

Example 11.1. Say alice wanted to send bob a movie (legally) and alice only has two options. Either pass the data to eve, who will in turn "give it" to bob, or reliably pass some data of max length l along a bandwidth'd channel straight to bob.

How can alice securely share the entire message? She would need a **hash function**, which serve a Block Cipher-esque purpose in data integrity. Let's see a definition now.

Definition (Hash function)**.** A **hash function** is an efficient algorithm $\mathbf{H}{:}\mathcal{M} \to \{0,1\}^l$ that maps messages in $\mathcal{M}$ to shorter l-bit strings, called **hashes** or **digests**, i.e., $|M| > 2^l$.

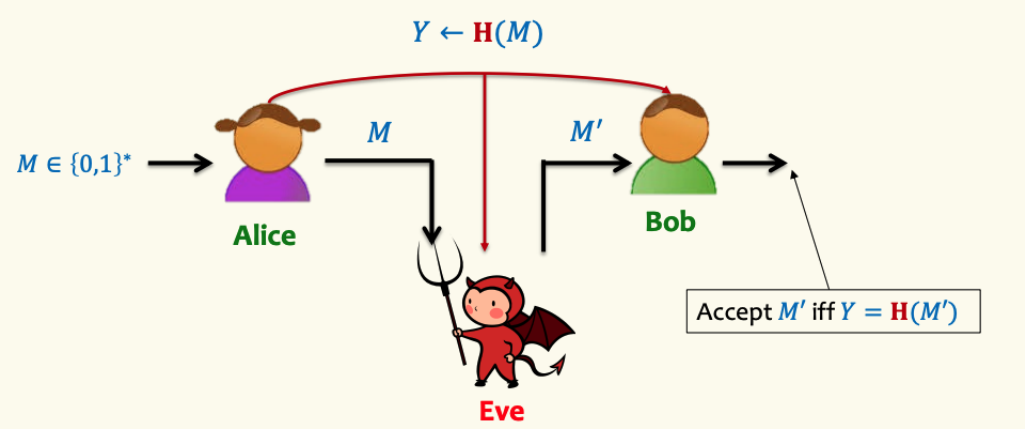
By the Pigeonhole principle, there exsts many messages in $\mathcal{M}$ mapped to the same digest Y- we call these the **preimages** of Y.

- On average, $|\mathcal{M}|/2^l$ preimages for each digest $Y \in \{0,1\}^l$
- Two distinct preimages of the same digest are called a **collision**
- Secure hash functions implies its *hard* to find collisions

Great, continuing with our alice-eve-bob story, how can alice and bob utilize this hash function? Well, alice can choose to pass along $Y \leftarrow \mathbf{H}(M)$ to bob via the secure l-bit channel.

Using this, bob can identify whether or not the message M' given to him by eve is correct by seeing if $Y = \mathbf{H}(M')$ as well.

Now, if we were eve, we'd want to come up with some $M' \neq M$ with the same hash. In other words, eve's goal is to find a collision.



Let's attempt to formalize this security goal. We want: "No feasible adversary can come up with a collision for a function $\mathbf{H} : \mathcal{M} \to \{0,1\}^l$".

But wait, take a step back. Is this even feasible? Think about it. These collisions *must* exist by pigeonhole, and if an adversary just hard-code in collisions, we cannot avoid it. Wat do ??

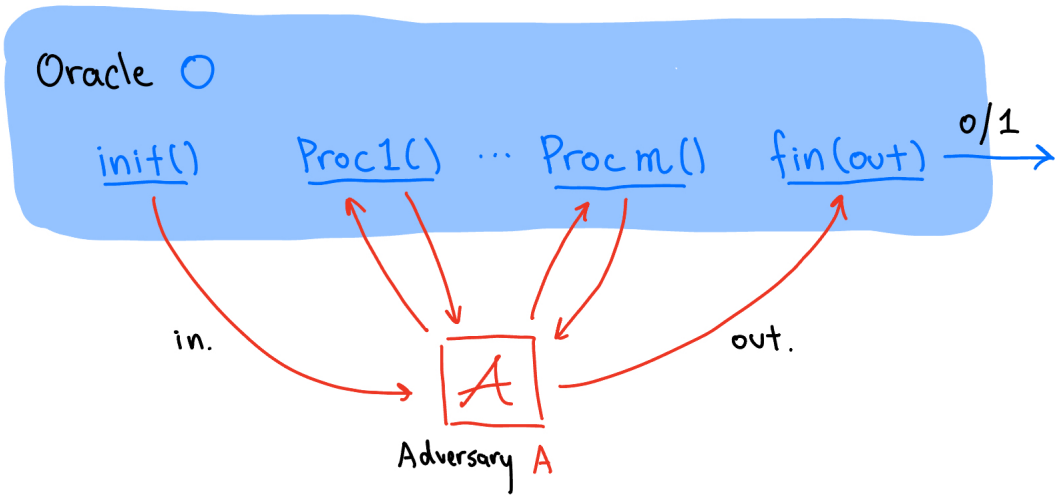
We define hash functions with **seed input**, i.e., $\mathbf{H} : \{0,1\}^s \times \mathcal{M} \to \{0,1\}^l$. Here, the seed is *not* a key, but rather a short, but public, parameter. With this, we can define a new security goal to be:

Given $S \xleftarrow{\$} \{0,1\}^s$, adversary cannot find $M_1 \neq M_2$ such that $\mathbf{H}(S, M_1) = \mathbf{H}(S, M_2)$

To formalize this notion, we first take a detour and reevaluate our definition of an oracle.

- So far, we've viewed "distinguishers" and "adversaries" as one and the same. They're not. Adversaries don't necessarily have to return a bit
  - Distinguishers = adversaries that output a bit
- We've also omitted one of the procedures of an oracle. The **Finalization Procedure** `Fin` takes in the adversary's output as input, and produces an output bit
  - It basically decides if the adversary has won (outputs 1 if so)
  - Previously, it's been implicit and just mirrored the output of a distinguisher

Intuition. *Here's a picture to truly visualize this new model.*



Now, $\Pr[A^O \Rightarrow 1]$ = probability that `Fin`'s output is 1.

Great, now that we've fixed up our definition of an oracle, let's define one.

Definition (Collision Resistance (CR))**.** We define Collision Resistance as

Algorithm 17 Oracle for Collision Resistance (CR)

```
Oracle CR[H]:

procedure INIT:
    S ←$ {0,1}^s
    return S

procedure FIN(M_1, M_2):
    if M_1 ≠ M_2 and H(S, M_1) = H(S, M_2) then
        return 1
    else
        return 0
```

From here, we define

$$\mathsf{Adv}^{\mathrm{cr}}_{\mathbf{H}}(A) = \Pr\left[A^{\mathrm{CR}[\mathrm{H}]} \Rightarrow 1\right]$$

H is (t, ϵ)-collision resistant if $\mathsf{Adv}^{\mathrm{cr}}_{\mathbf{H}}(A) \leq \epsilon$ for all A running in time at most t

H is **collision resistant** if $\mathsf{Adv}^{\mathrm{cr}}_{\mathbf{H}}(A)$ is negligible for all polynomial-time A

(running times typically parameterized by l)

We then took a look at a pretty generic example of an attack on collision resistance.

Example 11.2. This is a generic birthday attack.

Algorithm 18 Generic Birthday Attack

adversary $A^q_{\mathbf{Bday}}(S)$:
for $i = 1$ to q **do**
    $M_i \leftarrow\!\!\$\ \{0,1\}^n; Y_i \leftarrow \mathbf{H}(S, M_i)$
if $\exists i \neq j : M_i \neq M_j \wedge Y_i = Y_j$ **then**
    return (M_i, M_j)
else
    return $(0^n, 1^n)$

To find the advantage, we see that

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{cr}}_{\mathbf{H}}(A^q_{\mathrm{Bday}}) &\geq \Pr\left[\exists i \neq j : M_i \neq M_j \wedge Y_i = Y_j\right] \\
&\geq \Pr\left[\exists i \neq j : Y_i = Y_j\right] - \Pr\left[\exists i \neq j : M_i \neq M_j\right] \\
&\geq p_{\mathrm{coll}}(q, \{0,1\}^l) - p_{\mathrm{coll}}(q, \{0,1\}^n) \\
&\geq 1 - e^{-\frac{q(q-1)}{2^{l+1}}} - \frac{q(q-1)}{2^{n+1}}
\end{aligned}$$

So basically, if $q = 2^{l/2}$, and $n >> l$, then collision with probability ≥ 0.3. And this is why it's called a "birthday" attack- it utilizes the birthday paradox to produce a unexpectedly high collision probability.

Because of this birthday attack, the bottom line is that a good hash function should have $l \geq 256$, because we typically want security against time 2^{128} in practice.

The clever few may notice that this "birthday attack" actually can't be executed by eve in our example from previously, because M is chosen by alice, and eve can technically only choose M'.

So while collisio resistance guarantees that Eve cannot send an invalid message, the actual correct notion is weaker (and harder to break), called **2nd preimage resistance**.

Algorithm 19 2nd Preimage Resistance (2PR)

Oracle 2PR[**H**]:

procedure INIT:
  queried $\leftarrow 0$
  $S \leftarrow\!\!\$\ \{0,1\}^s$

procedure MSG(M):
  if queried $= 0$ **then**
    queried $\leftarrow 1$
    $M_1 \leftarrow M$
  return S

procedure FIN(M_2):
  if queried $= 1$ **and** $M_1 \neq M_2$ **and** $\mathbf{H}(S, M_1) = \mathbf{H}(S, M_2)$ **then**
    return 1
  else
    return 0

The advantage of **2PR** is that the seed is only returned after the first part of the collision is fixed. Now, this implies that

- asymptotically, **CR** implies **2PR**... theoretically it shouldn't be harder to break, but
- concretely, **2PR** should be harder to break than **CR**.
  - Birthday attacks break **CR** with any hash function in time at most $2^{l/2}$, where as
  - no generic attack against **2PR** in time better than 2^l.

12 Lecture 12: Oct. 20th

Today's lecture relied on a lot of intuition and examples, which I personally am a big fan of. We started with some applications of cryptographic hash functions in the real world, and then went into the construction of a typical hash function in Merkle-Damgård.

12.1 Applications of Hash Functions

We started with a recollection of hash functions and the collision resistance oracle. The important thing to remember is the newly introduced finalization procedure, as well as the 2pr oracle that prevents against a generic birthday attack.

We then did an application of hashing functions, in good-old seattle fashion:

Example 12.1. I want to convince you that I know, right now, whether the Mariners or Blue Jays will go to the world series, without revealing who will win today (it's game 7 today). How?

The naive solution could be:

1. Publish $\mathbf{H}(S, \text{winner}) = Y$ on EdStem today
2. Later on, you learn winner and check that $\mathbf{H}(S, \text{winner}) = Y$

But this presents a problem- because the seed is public, this doesn't hide the winner at all!

Another solution could be:

1. Pick (secret) $R \leftarrow\!\!\$\ \{0,1\}^n$
2. Publish $\mathbf{H}(S, R||\text{winner}) = Y$ on EdStem today
3. Later on, you learn winner, I reveal R, and check that $\mathbf{H}(S, R||\text{winner}) = Y$

This would work! Cheating would inherently require breaking collision resistance, but winner remains hidden without revealing R.

This example introduces us to what's called the **"Binding and Hiding Commitment"** property of hash functions.

Hash functions are also used for password protection.

Example 12.2. How should a web server store the clients' passwords? We COULD just have a simple sql table matching each client (some identifier) to their passwords.

Client 1	PW1
Client 2	PW2
Client 3	PW3

But this is a horrible idea. Once the table's stolen, everyone's cooked. So instead, we can utilize hash functions to build a different table, where

Client 1	$\mathbf{H}(S,\text{PW1})$
Client 2	$\mathbf{H}(S,\text{PW2})$
Client 3	$\mathbf{H}(S,\text{PW3})$

This gives a couple advantages.

1. Most importantly, if the table is stolen, it's still hard to find anyone's password, since that would require breaking the hash (pre-image resistance)
2. But also, hashing two distinct passwords is unlikely to collide (collision resistance)

Before moving onto contsruction of hash functions, a quick intuitive summary.

> **Intuition.** Pre-image resistance *prevents anyone from "breaking" the hash function and reverse engineering to recover the original plaintext.*
>
> Collision resistance *prevents anyone from finding a different plaintext that produces the same hash*
>
> *A "good" hash function should do both.*

12.2 Construction of Hash Functions

And finally, we ended lecture with a discussion on the construction of hash functions. Analogously, these hash functions are similar in concept to different encryption modes (like CBC or CTR) that we've discussed in the past, but obviously for hashing rather than encrypting.

And much like how encryption modes had *block ciphers*, hash functions also has a building block called **compression functions h** : $\{0,1\}^s \times \{0,1\}^{l+1} \to \{0,1\}^l$ that is able to compress an $(l+1)$-length input into an l-length output with the help of some seed.

Now, we introduce a hash function construction known as the **Merkle-Damgård**. Formally, $\mathsf{MD}^{\mathbf{h}}_n : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^l$

Algorithm 20 Merkle-Damgård

procedure $\mathsf{MD}^{\mathbf{h}}_n(S, M)$:
  Parse $M = m_1...m_n$, where each m_i is 1 bit
  $V_0 \leftarrow 0^l$
  for $i = 1, ..., n$ **do**
    $V_i \leftarrow \mathbf{h}(S, V_{i-1}||m_i)$
  return $Y \leftarrow V_n$

> **Intuition.** *This is kinda similar in concept to CBC... i guess? Using the previous hash and append a current bit, find the new hash.*

Merkle-Damgård has an important property.

Theorem 12.1. *If $\boldsymbol{h}$ is collision resistant, then $\mathsf{MD}_n^{\boldsymbol{h}}$ is collision resistant.*

But what if we don't want to restrict messages to just length n but rather an arbitrary length? I.e., $\mathcal{M} = \{0,1\}^*$?

Naively, we could just like repeat for $|M|$ times rather than n. But that's actually bad and insecure. Intuitively, something like $\mathsf{MD}_*^{\mathbf{h}}(S, 0) = \mathsf{MD}_*^{\mathbf{h}}(s, 00) = \cdots = 0^l$ could happen.

So to do this, we still need to somehow "fix" the length of our input.

Algorithm 21 Merkle-Damgård

procedure $\mathsf{MD}_*^{\mathbf{h}}(S, M)$:
  $m_n ... m_1 \leftarrow \text{encode}(M)$
  return $\mathsf{MD}_n^{\mathbf{h}}(S, m_n ... m_1)$

This 'encode' function can be arbitrary and black-box, all we would require from it is a single property: **suffix freeness**.

Definition (Suffix freeness of encoding)**.** For every $M \neq M'$, with

$$\text{encode}(M) = m_n ... m_1$$
$$\text{encode}(M') = m'_n ... m'_1$$

there exists $i \in \{1, ..., \min(n, n')\}$ such that $m_i \neq m'_i$

13 Lecture 13: Oct. 22nd

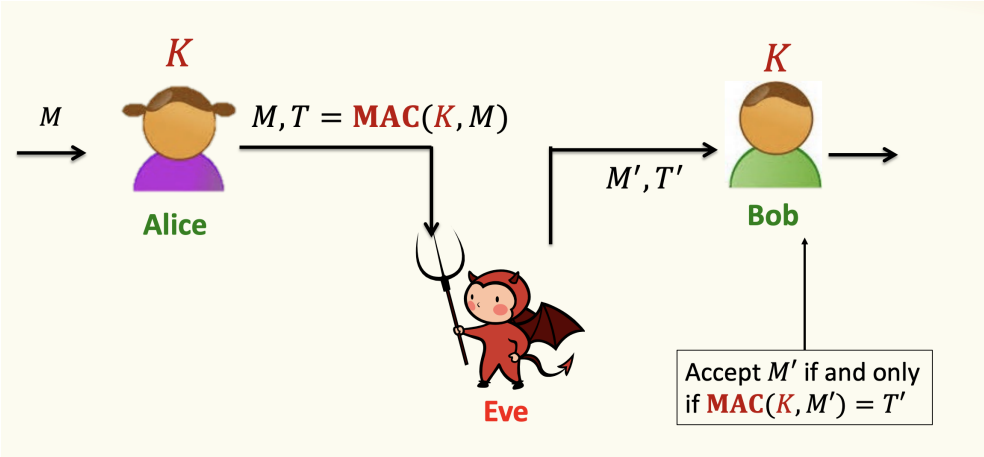
Today in lecture we discussed a new type of cryptographic object- MACs! In fact, before going into its applications and encryptions, we first discussed its construction, which is a cool synthesis of some of the previous topics we've seen in the class. Not gonna lie, I did not follow very much during the lecture, but after reviewing the contents, I feel rather comfortable with this.

13.1 MACs

We began lecture with some foreshadowing- we are going to be introducing a new cryptographic object today. And as usual, it will be built up from intuition into a formalized oracle.

But first, we started with a recollection of the idea of passing information over some unreliable channel, and why a hash was needed. But now, what if even that hash channel didn't exist, and instead, alice and bob both had access to some secret key? Would it still be possible to pass along a message without eve knowing?

Actually, yes! Here is where we introduced a **Message Authentication Code (MAC)**.



Essentially, a **message-authentication code (MAC)** is an efficient algorithm MAC : $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$.

The security expectation we're concerned with here is **Unforgeability under a Chosen-Message Attack (UF-CMA)**.

- Even if the adversary gets to learn $\mathbf{MAC}(K, M_i)$ for chosen messages $M_1, M_2, ...$, the adversary cannot predict $\mathbf{MAC}(K, M^*)$ for a *new* message $M* \notin \{M_1, M_2, ...\}$.
- We can define this formally within our oracle framework

Definition (UF-CMA security)**.** Formally, we define UF-CMA security as

Algorithm 22 UF-CMA Oracle

```
oracle UF − CMA[MAC]

procedure Init:
    Q → ∅
    K ←$ {0,1}^k

procedure Eval(M):
    Q ← Q ∪ {M}
    return MAC(K, M)

procedure Fin(M*, T*):
    if M* ∉ Q and MAC(K, M*) = T* then
        return 1
    else
        return 0
```

The advantage is defined as

$$\mathsf{Adv}^{\text{uf-cma}}_{\mathbf{MAC}}(A) = \Pr[A^{\mathsf{UF-CMA}[\mathbf{MAC}]} \Rightarrow 1]$$

$\mathbf{MAC}$ is (t, ϵ)-$\mathsf{UF-CMA}$ secure if $\mathsf{Adv}^{\text{uf-cma}}_{\mathbf{MAC}}(A) \leq \epsilon$ for all A running in time at most t.

$\mathbf{MAC}$ is $\mathsf{UF-CMA}$ secure if $\mathsf{Adv}^{\text{uf-cma}}_{\mathbf{MAC}}(A)$ is negligible for all polynomial-time A.

Running time is typically parameterized by k.

We then went onto a discussion how exactly to build a "secure" mac. In fact, we made a very interesting connection to a topic discussed a few lectures prior.

But first, we need to have a quick discussion about PRFs. Thus far, we've only concerned ourselves with prfs with length exactly n, but what if we wanted arbitrary length PRFs?

It turns out, the notion of PRFs extend very naturally to **variable-input length** (VIL) PRFs, where the notion of advantage is still, exactly the same! We'll discuss how to construct these in a little bit, but first:

Theorem 13.1 (PRFs are UF-CMA Secure MACs). *If $\boldsymbol{F} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ is a (t, ϵ)-PRF, then it is also (t', δ)-UF-CMA secure for $\delta = \epsilon + \frac{1}{2^l}$ and $t' \approx t$*

Corollary 13.1.1. *If $\boldsymbol{F}$ is a PRF, then it is also UF-CMA secure if $l = \omega(\log k)$*

As for the proof, the key step we need is to find some oracle O such that

1. $\mathsf{O}^{\mathsf{KF}[\mathbf{F}]} \equiv \mathsf{UF}-\mathsf{CMA}[\mathbf{F}]$
2. $\Pr\left[A^{\mathsf{O}^{\mathsf{KF}[\mathbf{F}]}} \Rightarrow 1\right] \leq \frac{1}{2^l}$ for all A.

From here, we can carry out the sequence of operations here

$$\begin{aligned}
\mathsf{Adv}^{\text{uf-cma}}_{\mathbf{F}}(A) &= \Pr[A^{\mathsf{O}^{\mathsf{KF}[\mathbf{F}]}} \Rightarrow 1] \\
&= \Pr[A^{\mathsf{O}^{\mathsf{KF}[\mathbf{F}]}} \Rightarrow 1] - \Pr[A^{\mathsf{O}^{\mathsf{RF}[*,l]}} \Rightarrow 1] + \Pr[A^{\mathsf{O}^{\mathsf{RF}[*,l]}} \Rightarrow 1] \\
&\leq \left|\Pr[A^{\mathsf{O}^{\mathsf{KF}[\mathbf{F}]}} \Rightarrow 1] - \Pr[A^{\mathsf{O}^{\mathsf{RF}[*,l]}} \Rightarrow 1]\right| + \frac{1}{2^l} \\
&= \mathsf{Adv}^{\text{prf}}_{\mathbf{F}}(A^{\mathsf{O}}) + \frac{1}{2^l} \\
&\leq \epsilon + \frac{1}{2^l}
\end{aligned}$$

Now, consider the following oracle

Algorithm 23 Oracle $\mathsf{O}^{\mathsf{O}'}$:

```
1:  procedure INIT
2:
3:  procedure EVAL(M):
4:      Q ← Q ∪ {M}
5:      return O'.Eval(M)
6:
7:  procedure FIN(M*, T*):
8:      if M* ∉ Q and O'.Eval(M*) = T* then
9:          return 1
10:     else
11:         return 0
```

Consider the following cases:

Case 1: $\mathsf{O}' = \mathsf{KF}[\mathbf{F}]$. Then, we'd have $\mathsf{O}'.\mathsf{Eval}(M) = \mathbf{F}(K, M)$ for $\mathsf{KF}[\mathbf{F}]$'s key K, which means in this case

$$\mathsf{O}^{\mathsf{KF}[\mathbf{F}]} \equiv \mathsf{UF}-\mathsf{CMA}[\mathbf{F}]$$

Case 2: $\mathsf{O}' = \mathsf{RF}[*, l]$. Notice here that in Fin, M^* is always new because $M^* \notin Q$, meaning that $\mathsf{O}'.\mathsf{Eval}(M^*)$ is uniforma and will equal T^* with probably exactly $1/2^l$. In this case

$$\Pr[A^{\mathsf{O}^{\mathsf{RF}[*,l]}} \to 1] \leq \frac{1}{2^l}$$

These cases satisfy our proof, and we've reached the desired results!

Another important corollary is that

Corollary 13.1.2. *If $\boldsymbol{F}$ is UF-CMA secure, then it not necessarily a PRF.*

This can be easily shown by just attaching a 0 at the end of some UF-CMA secure $\mathbf{F}$, and noticing that it still is UF-CMA secure, but is no longer pseudorandom.

Great, now that we know a secure MAC is basically just some (VIL) PRF, how exactly do we construct one given the "arbitrary length" requirement?

One way is to actually cleverly apply two previous concepts. Recall:

- A good *block cipher* $\mathbf{E}$ should be a length n prf. We don't need to reinvent the wheel here.
- A good *hash function* $\mathbf{H}$ should be able to compress an arbitrary length string to some fixed length, while avoiding collisions.

This brings us to

Theorem 13.2. $\boldsymbol{F}((K,S),X) = \boldsymbol{E}(K, \boldsymbol{H}(S,X))$ *is a VIL-PRF.*

Wait wait wait wait a sec... if all we need is some length compression, why can't we JUST use a hash function? Why bother with all of this block cipher prf bs??

In other words, is $\mathsf{MAC}(K,M) = \mathbf{H}(S, K||M)$ UF-CMA Secure? Well... no. There is an attack known as the **length-extension attack** that makes $\mathbf{H}(S,K||M)$ insecure.

- Roughly speaking: given $T = \mathbf{H}(S,X)$ we can efficiently compute $\mathbf{H}(S, X||Y)$ from T, S, and Y without actually knowing X
- For example, The Merkle-Damgård hash functions are subject to these types of attacks, and thus do not give good MACs!

14 Lecture 14: Oct. 24th

HOLY GIGA LECTURE this was the longest lecture notes yet. Today we basically concluded our discussion on symmetric encryption schemes with secret keys (the first half of the course, which makes sense because midterm next week jumpscare ahhhhhhh!!!!). This was a very technically challenging lecture, packed with hella yap but also hella notation. But imo it was very satisfying to see how the puzzles of the past 5 weeks fit together to produce the golden standard of authenticated encryption.

14.1 Authenticated Encryption

To begin lecture, the professor contextualized that today we will be discussing *authenticated encryption*, which is the golden standard of security when it comes to symmetric encryptions that share a private key, which is exactly what we've covered thus far in the course. This should be a good "wrap-up" of all the topics we've discussed.

But before that, we began with a brief recollection of the previous lecture. Specifically, we emphasized that using a hash function like $\mathbf{H}(S, K||M)$ as MACs is not a good approach as many constructions of hash functions are vulnerable to length-extension attacks. Do note here though, that *this doesn't mean that* $\boldsymbol{H}$ *is a bad hash function*, it simply means that it would make a bad MAC.

Ironically, enough the most common MAC construction is called an HMAC, which is a construction of a "pseudo-hash function" that works as a MAC.

We then followed up with a discussion about ciphertext integrity (denoted INT-CTXT). We came up with a "new" security goal:

> "It must be infeasible to produce a new ciphertext which decrypts to a valid plaintext, even after seeing large numbers of ciphertexts for the same key"

And so we say that a secret key symmetric encryption scheme satisfies **Authenticated Encryption** if it is both IND-CPA secure and INT-CTXT secure.

Thus far, we haven't see much of data integrity. Think about it. CBC doesn't satisfy this because we've shown how a padding oracle attack can reveal much information. Similarly, CTR definitely doesn't satisfy this because every ciphertext will decrypt to some plaintext by construction.

Don't believe me? Let's take a look at an example of an attack on CTR. This is known as the **malleability attack**.

Example 14.1. Imagine eve sees the following ciphertext C encrypted with CTR, and decides to change C to C' and gives it to bob.

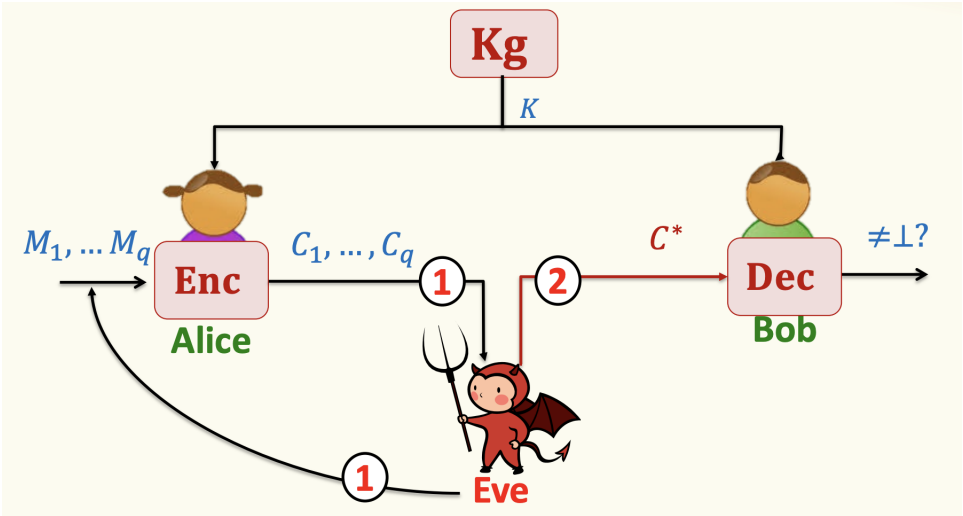
What does this change? Well since we know CTR is some mask, we know that this will specifically change the 12th character of the plaintext to something else upon decrypting.

So clearly, CTR does NOT uphold integrity here. This motivates us to maybe provide an updated version of a very old and familiar definition.

A **symmetric encryption scheme** is a triple of algorithms $\Pi = (\mathbf{Kg}, \mathbf{Enc}, \mathbf{Dec})$, where:

- The (*randomized*) key-generation algorithm **Kg** takes no input and outputs a key K.
- The encryption algorithm **Enc** takes the key k and some plaintext M, and outputs a ciphertext $C \leftarrow \mathbf{Enc}(K, M)$
- The decryption algorithm **Dec** is such that $\mathbf{Dec}(K, \mathbf{Enc}(K, M)) = M$ for every plaintext M and key K output by **Kg**.
  - **NEW**: We allow **Dec** to also return an error symbol $\perp$

Let's look at a visual intuition, and then go into an oracle formalization (as per usual). For the intuition, let's once again revisit a familiar graphic.



The idea of INT-CTXT as a requirement of symmetric encryption schemes is that, whatever ciphertexts alice sends to eve, eve SHOULD NOT be able to learn to generate an "understandable" ciphertext C^* to bob such that decryption doesn't produce $\perp$.

> **Intuition.** *By this graphic, it sure looks like INT-CTXT has a pretty similar idea to UF-CMA which we discussed under the context of MACs. It indeed is similar, but remember that INT-CTXT is the property of an encryption scheme, so the syntax of the cryptographic objects we are dealing with are different.*

More formally, let's introduce the oracle.

Algorithm 24 Oracle $\mathsf{INT-CTXT}[\Pi]$:

procedure INIT:
    $Q \leftarrow \emptyset$
    $K \leftarrow\$ \mathbf{Kg()}$

procedure ENCRYPT(M):
    $C \leftarrow\$ \mathbf{Enc}(K, M)$
    $Q \leftarrow Q \cup \{C\}$

procedure FIN(C^*):
    if $C^* \notin Q$ **and** $\mathbf{Dec}(K, C^*) \neq \perp$ **then**
        return 1
    else
        return 0

The advantage is defined as

$$\mathsf{Adv}^{\text{int-ctxt}}_{\Pi}(A) = \Pr[A^{\mathsf{INT-CTXT}[\Pi]} \Rightarrow 1]$$

Π is (t, ϵ)-$\mathsf{INT-CTXT}$ secure if $\mathsf{Adv}^{\text{int-ctxt}}_{\Pi}(A) \leq \epsilon$ for all A running in time at most t.

Π is $\mathsf{INT-CTXT}$ secure if $\mathsf{Adv}^{\text{int-ctxt}}_{\Pi}(A)$ is negligible for all polynomial-time A.

Running time is typically parameterized by key length.

Like we've stated above with the intuition block, but since we've established a similar notion of UF-CMA secure when we discussed MACs, we can kind of use that to our advantage.

In other words, we would ideally like to build an authenticated encryption scheme by combining

- An IND-CPA encryption scheme Π

- A UF-CMA secure $\mathbf{MAC} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$

and the result should satisfy IND-CPA and INT-CTXT security!

There are two general trains-of-thought here, aptly named "encrypt-then-mac" and "mac-then-encrypt".

Algorithm 25 Encrypt-then-Mac

```
1: procedure EtM.Enc(K_1||K_2, M):
2:     C' ←$ Enc(K_1, M)
3:     T ← MAC(K_2, C')
4:     return C = C'||T
5:
6: procedure EtM.Dec(K_1||K_2, C):
7:     Parse C = C'||T
8:     if T = MAC(K_2, C') then
9:         M ← Dec(K_1, C')
10:        return M
11:    else
12:        return ⊥
```

Algorithm 26 Mac-then-Encrypt

```
1: procedure MtE.Enc(K_1||K_2, M):
2:     T ← MAC(K_2, C)
3:     C ←$ Enc(K_1, M||T)
4:     return C
5:
6: procedure MtE.Dec(K_1||K_2, C):
7:     M' ← Dec(K_1, C)
8:     if M' ≠⊥ then
9:         Parse M' as M||T
10:        if MAC(K_2, M) = T then
11:            return M
12:    else
13:        return ⊥
```

Note here that both of these approaches should be IND-CPA secure, although we are not going to spend the time in lecture to prove it (very doable though). So security? Check.

But a note on that note is that despite both being IND-CPA secure, Encrypt-then-Mac here hides only the information of M, whereas Mac-then-Encrypt hides information of both M and T.

Okay, great. Now how about integrity? Well, it turns out that Encrypt-then-Mac ensures complete ciphertext integrity, whereas Mac-then-Encrypt ensures a slightly weaker version of that called plaintext integrity (INT-PTXT).

Algorithm 27 Oracle $\mathsf{INT-PTXT}[\Pi]$:

```
procedure INIT:
    Q ← ∅
    K ←$ Kg()

procedure ENCRYPT(M):
    C ←$ Enc(K, M)
    Q ← Q ∪ {M}

procedure FIN(C*):
    if Dec(K, C*) ∉ Q then
        return 1
    else
        return 0
```

The advantage is defined as

$$\mathsf{Adv}^{\text{int-ptxt}}_{\Pi}(A) = \Pr[A^{\mathsf{INT-PTXT}[\Pi]} \Rightarrow 1]$$

Π is (t, ϵ)-$\mathsf{INT-ptxt}$ secure if $\mathsf{Adv}^{\text{int-ptxt}}_{\Pi}(A) \leq \epsilon$ for all A running in time at most t.

Π is $\mathsf{INT-PTXT}$ secure if $\mathsf{Adv}^{\text{int-ptxt}}_{\Pi}(A)$ is negligible for all polynomial-time A.

Running time is typically parameterized by key length.

> **Intuition.** *Essentially, plaintext integrity is a weaker notion in that, an adversary eve, by analyzing ciphertexts, is able to produce a new ciphertext that bob will be able to decrypt. This sounds bad, but the caveat here is that the new ciphertext will guarantee to be decrypted to a message that was originally sent by alice.*

In most applications, INT-PTXT is enough because all we care about is that Eve should not be able to inject new fake messages. That said thoough, when given the option, **INT-CTXT should be favored!** (duh)

One quick note at the end of the lecture: EtM and MtE schemes are still not perfect! This gets quite advanced, but intuitively, an adversary can measure the time at which execution of the program stops to determine at which stage in the program their input failed.

- For example, for MtE, padding errors cause execution to stop after time **Dec**

- whereas MAC errors cause execution to stop after time **Dec** + time **MAC**

This is just yet another way for some adversary to take advantage of our program, just to keep in mind for the future. However, for the sake and scope of this class, we will abstract that away and assume that standard threat models (IND-CPA, INT-CTXT) does not give adversary timing information.

15 Lecture 16: Oct. 31st (boo!)

Today in lecture we completely shifted our focus in cryptography. Personally I think I will enjoy this type of cryptography much more, just because I enjoy the math aspect of it. At first the idea of key exchange was a little foreign and confusing to me. This was also my first formal look at the definition of a group despite being a math person lol.

15.1 Key exchange

We are back! The past two lectures were dedicated to both a midterm review (Oct. 27th) and the actual midterm (Oct. 29th).

Now that the first half of the course has concluded, we will be switching our perspective on cryptography, and will now be looking at mathematical cryptography for once! The backstory is, in 1976, these two dudes from Stanford was like

> "yeah key generation is lowkey really inefficient and getting really expensive now that the messages we encrypt are longer, soooo... let's introduce *public key cryptography*!"

(real quote trust me bro)

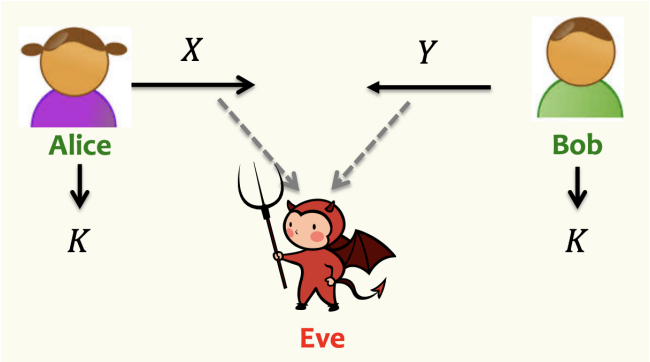
To visualize this idea, let's see an example with our good friends alice bob and eve.

Example 15.1. Suppose once again that Alice and Bob are trying to communicate through some shared channel. Much like usual, Eve is observing and spying on this communication channel (let's not worry about message tampering for now).

But unlike before, Alice and Bob did *not* start with a predetermined secret key. In fact, their goal now is to communicate in such a way that they can **agree on** a shared **key** K.

The idea of key exchange is that, regardless of the messages exchanged between Alice and Bob, the final key K should appear pseudorandom to Eve. In other words, from Eve's perspective, even after seeing all exchanged messages, the distribution of (X, Y, K) should be computationally indistinguishable from (X, Y, K'), where K' is an independent random key:

$$\{X, Y, K\} \approx \{X, Y, K'\}$$



Prior to today, when we discussed symmetric key cryptography, the computationally hard problems were always "custom-made", whether that's modern block ciphers like AES or hash functions like SHA3.

But now, in public key cryptography, we will instead be taking advantage of mathematically difficult problems, which will enable us to be able to perform actions like key exchange. But before we get into the technicality of cryptography, maybe we should learn some math first LETS GO!!

15.2 Algebra & Number theory

Definition (Groups)**.** A **group** $(\mathbb{G}; *)$ is a set of elements $\mathbb{G}$ and a binary operation $*$ with the following properties:

1. **Closure**: $\forall a, b \in \mathbb{G} : a * b \in \mathbb{G}$
2. **Identity**: $\exists 1$ such that $\forall a \in \mathbb{G}$, $1 * a = a * 1 = a$
3. **Associativity**: $\forall a, b, c \in \mathbb{G}$, $(a * b) * c = a * (b * c)$
4. **Inverse**: $\forall a \in \mathbb{G} : \exists b \in \mathbb{G}$ such that $a * b = b * a = 1$.
   Here, we denote b as the inverse of a, $b = a^{-1}$

Remark. There are a few remarks here:

- Identity is unique
- inverses are unique for each element
- Groups are not necessarily commutative. If commutativity holds, then the group is called **abelian**.

In the discussion within this class, we will mostly be dealing with *finite* groups (i.e. $|\mathbb{G}|$ is finite. For example, we'll say that a problem is computationally hard if it's in time $O(|\mathbb{G}|)$, where as easy is in time $O(\log |\mathbb{G}|)$.

Theorem 15.1 (Division with remainder). *For any a, N with $N > 0$, there exists unique q and r such that*

$$a = Nq + r \text{ and } 0 \leq r < N$$

We denote r as a mod N

Definition (Congruence classes)**.** We define $\mathbb{Z}_N = \{0, 1, ..., N-1\}$. Additionally, we define $+_N : \mathbb{Z}_N \times \mathbb{Z}_N \to \mathbb{Z}_N$ such that

$$a +_N b = (a + b) \bmod N$$

Now, let's think. Is $(\mathbb{Z}_N; +_N)$ a group?

- Closure is trivial by definition of $+_N$
- Associativity is also trivial
- Identity is defined as 0. $a +_N 0 = 0 +_N a = a$
- Inverses is defined as $(-a)$, where $-a = N - a$.

Bang. It's a group. This was pretty simple, so maybe let's think about multiplication.

We define **modular multiplication** $\times_N : \mathbb{Z}_N \times \mathbb{Z}_N \to \mathbb{Z}_N$ such that

$$a \times_N b = (a \cdot b) \bmod N$$

Is $(\mathbb{Z}_N; \times_N)$ a group?

- Closure and associativity are trivial by definition of $\times_N$
- Identity is defined as 1. $a \times_N 1 = 1 \times_N a = a$
- Inverses? Take a look at $\mathbb{Z}_6$. What's the inverse of 4?
  - $4 \times_6 0 = 0$, $4 \times_6 1 = 4$, $4 \times_6 2 = 2$, $4 \times_6 3 = 0$, $4 \times_6 4 = 4$, $4 \times_6 5 = 2$
  - No inverses!

If a is invertible, then $\exists k, b = a^{-1}, ab = kN + 1$, or equivalently

$$ab - kN = 1$$

But how do we know when k, b exists? Before we answer this question, let's recall the definition

Definition (Greatest common divisor)**.** $\gcd(a, N)$ is the greatest positive integer that divides both a and N.

And actually, there is a cool lemma.

Theorem 15.2 (Bezout's theorem). *There exists $b, k \in \mathbb{Z}$ such that $ab - kN = 1$ if and only if $\gcd(a, N) = 1$. Furthermore, b, k can be found using the extended Euclidean algorithm*

Okay, now that we know what makes a number invertible in $\mathbb{Z}_N$, why don't we build a new set using elements from $\mathbb{Z}_N$ so that we can actually form a group?

Definition ($\mathbb{Z}_N^*$)**.** For every $n \geq 1$, define

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$$

And so now, every element of $\mathbb{Z}_N^*$ should be invertible! We thus conclude that $\mathbb{Z}_N^*$ **is a group.**

Continuing on, we then defined a helpful function that helped us find the size of any $\mathbb{Z}_N^*$.

Definition (Euler Totient function)**.** For $N \in \mathbb{Z}_{>0}$, define

$$\varphi(N) := |\{a : 1 \leq a \leq N, \gcd(a, N) = 1\}|$$

Example 15.2. For any prime p, we have $\varphi(p) = p - 1$.
For any prime p, q, we have $\varphi(pq) = pq - p - q + 1 = (p - 1)(q - 1)$
In fact, for general $N = \Pi p_i^{k_i}$, we have $\varphi(N) = \Pi p_i^{k_i - 1}(p_i - 1)$

16 Lecture 17: Nov. 3rd

Today in lecture we continued with our discussion on group theory and algebra, specifically with the cryptographic goal of finding a computationally hard problem in mind. But tbh, this lecture was honestly just hella theorems and definitions being spammed at us. It set the ground work for a lot more operations and algorithms we will explore in the future, but for now, it's just time to review these and make sure I have my fundamentals good.

16.1 Algebra & Number theory (cont'd)

We started off the lecture with a basic review on group theory that we've covered thus far. Specifically, recall the definition of groups, modular arithmetic (and its efficiency), properties of $\mathbb{Z}_N^*$.

More specifically, recall that we are learning all of this math in seeking of a computationally "hard" problem, and that most operations we've covered so far in $\mathbb{Z}_N/\mathbb{Z}_N^*$ can be efficiently computed (with worst-case $O((\log N)^2)$.

We then moved onto the concept of *group exponentiation*.

Definition (Group exponentiation)**.** Let $(\mathbb{G}, *)$ be a group. Given $x \in \mathbb{G}, e \in \mathbb{N}, e \geq 0$, we define $x^e \in \mathbb{G}$ as

$$x^e = \underbrace{x * x * \cdots * x}_{\text{e times}} \qquad x^0 = 1$$

There are a couple ways of computing this algorithmically. Naively, we have the following algorithm.

Algorithm 28 NaiveExp(x, e):

```
z ← 1
for q = 1 to e do
    z ← z * x
return z
```

The issue with this algorithm is that it's "slow", require $\Theta(e)$ group operations.

But wait, this might be a good thing! Recall that the goal of us exploring these mathematics is in seeking of a "hard" problem computationally so we can use it to our advantage when encrypting messages in cryptography.

Since this is a "slow" algorithm, did we just find a hard enough problem? Let's take a look at another algorithm that computes this:

Algorithm 29 SquareAndMult(x, e):

```
e = e_k e_{k-1} ... e_0 // binary representation of e
z ← 1
for i = k down to 0 do
    z ← z^2
    if e_i = 1 then
        z ← z * x
return z
```

Here, notice that since we're computing the *binary representation*, we will have at most $O(\log_2(e))$ group operations.

Okay, so group exponentiation wasn't it. It's too easy.

Let's maybe take a look at this same thought, but under the scope of modular exponetiation. More formally perhaps, we take some element $x \in \mathbb{Z}_N^*$, and define the set $\langle x \rangle$ to be

$$\langle x \rangle = \{x^e \bmod N \mid e \in \mathbb{N}\}$$

What does this look like?

Example 16.1. In $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$, $x = 3$, we have

$e =$	1	2	3	4	5	6
	3	2	6	4	5	1
$e =$	7	8	9	10	11	12
	3	2	6	4	5	1

Observation: the value of x^e *must* repeat! Albeit at different frequencies depending on the value of N.

To formalize this thought, let's introduce a theorem.

Theorem 16.1 (Euler's theorem). *For all $x \in \mathbb{Z}_N^*$, we have $x^{\varphi(N)} \equiv 1 \pmod N$*

Corollary 16.1.1. *For all $x \in \mathbb{Z}_N^*, e \geq 0$, we have $x^e \equiv x^{e \bmod \varphi(N)} \bmod N$.*

> **Intuition.** *Because of this, element in the group $\mathbb{Z}_N^*$ must be eventually repeated because (1) pigeonhole and (2) we can simply have $a(x^{\varphi(N)})$ which gives us a. This is a generalization of* Fermat's Little Theorem.

In fact, this idea can be extended to any group $\mathbb{G}$ beyond just $\mathbb{Z}_N^*$. This is exactly what LaGrange did. Bro basically came over to Euler's and said "this can actually be generalized!" So we have

Theorem 16.2 (LaGrange's Theorem). *For all $x \in \mathbb{G}$, we have $x^{|\mathbb{G}|} = 1$.*

Corollary 16.2.1. *For all $x \in \mathbb{G}, e \geq 0$, we have $x^e \equiv x^{e \ mod \ |\mathbb{G}|}$.*

(Technically this isn't the entirety of LaGrange's theorem, but rather just a corollary of it.)

Let's now explore the idea of $\langle x \rangle$ a little further, by exploring the idea of a "sub-group".

Fact. For any group $(\mathbb{G}, *)$, we have that for any $a \in \mathbb{G}$, the pair $(\langle a \rangle, *)$ is a group (and a **sub-group** of $\mathbb{G}$).

- **Closure** follows from the definition of $\langle a \rangle$.
- **Associativity** follows from $\mathbb{G}$
- **Identity** is also confirmed previously, since $1 \in \langle a \rangle$
- **Inverses** are also well-defined.

$$(x^e)^{-1} = x^{-e \bmod |\mathbb{G}|}$$
$$x^e * x^{-e \bmod |\mathbb{G}|} = x^e * (x^e)^{-1} = x^0 = 1$$

And now... more definitions! Since we have the idea of a sub-groups denoted with $\langle x \rangle$, we can explore a lot more interesting properties of them in seek of a hard compute.

Definition (Generator)**.** g is a **generator** of $\mathbb{G}$ if

$$\langle g \rangle = \{g^0 = 1, g^1, ..., g^{|\mathbb{G}|-1}\} = \mathbb{G}$$

Definition (Cyclic groups)**.** A group $\mathbb{G}$ is **cyclic** if it has a generator.

Example 16.2. Is $(\mathbb{Z}_N, +_N)$ cyclic for any $N \geq 1$?

Yes! The generator is 1:

$$\langle 1 \rangle = \{0, 1, 2, 3, ..., n-1\}$$

because the operation is addition!

Example 16.3. Is $(\mathbb{Z}_N^*, \times_N)$ cyclic for any $N \geq 1$?

It depends. For a group like $\mathbb{Z}_{13}^*$, we have

$$\langle 2 \rangle = \{1, 2, 4, 8, 3, 6, 12, 10, 7, 8, 11, 9\}$$

However, no generators exist for $\mathbb{Z}_{15}^*$. This actually gives us an interesting lemma.

Lemma 16.3. *The group $\mathbb{Z}_N^*$ is cyclic if and only if $N = 2, 4, p^k, 2p^k$, where $p > 2$ is an odd number.*

And finally, to end off the lecture, we discussed an operation that (we were told) will be *fundamental* for the forseeable future.

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order N. We now define 2 functions: **exponentiation** and **discrete log**.

$$\mathsf{Exp} : \mathbb{Z}_N \to \mathbb{G}, \qquad \mathsf{Exp}(x) = g^x$$
$$\mathsf{DLog} : \mathbb{G} \to \mathbb{Z}_N, \qquad \mathsf{DLog}(a) = x \text{ s.t. } g^x = a$$

With the `SquareAndMult()` algorithm we discussed earlier, we know that Exp is "easy", and can be calculated with $O(\log N)$ group ops. Our task now will be more focused on DLog, and specifically, under what conditions will DLog be "hard"?

Example 16.4. Given $\mathbb{Z}_{17}^* = \langle g \rangle$ for $g = 3$, and some $y \in \mathbb{Z}_{17}^*$, how do we find $x \in \mathbb{Z}_N = \mathbb{Z}_{16}$ such that $g^x = y$?

Well, naively, since $\mathbb{Z}_{17}^*$ is given to be cyclic, we can evaluate g^x on all possible x until successful. But this would be $\Theta(N)$.

We will see next lecture that we can, in fact, do better. But not much better.

17 Lecture 18: Nov. 5th

Today's lecture was HELLA content packed. We are finally relating the algebra and group thery that we've discussed in the past 2 lectures back to cryptographic concepts, thorugh the introduction of the Diffie-Hellman Key Exchange. This lecture we explored a lot of both mathematical examples, as well as intuition on the security regarding DHKE (with *three* different hardness / security measures!) There definitely is a lot of intuition being built up now through connecting math and computational hardness with cryptographic security.

17.1 Diffie-Hellman Key Exchange

To begin lecture today, we had a quick recollection of cyclic groups and discrete logs. We are now going to be applying these to a cryptographic context through the Diffie-Hellman Key Exchange (DHKE).

Recall the goal of *key exchange*: Alice and Bob both start with nothing, and have to agree on a key that must be secret and random to Eve, who has access to the channel of information flowing between Alice and Bob.

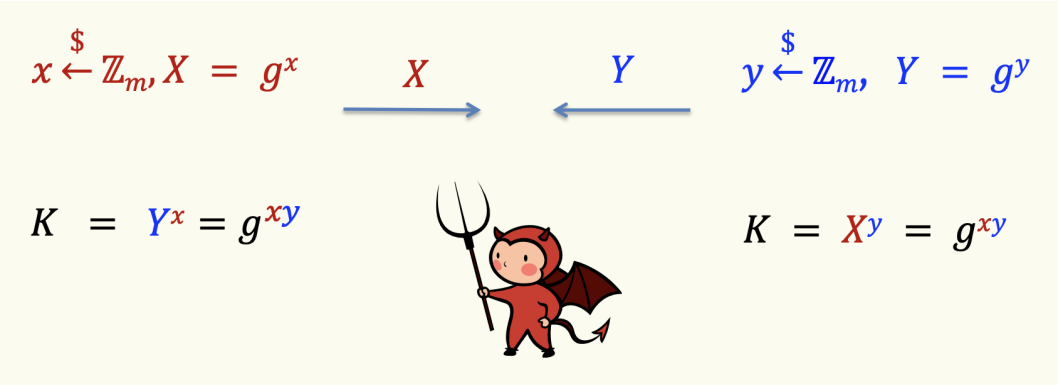
We now define DHKE. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group with order $|\mathbb{G}| = m$.

From alice's perspective, she will randomly sample $x \leftarrow\!\!\$\ \mathbb{Z}_m$, then perform group exponentiation to find $X = g^x$. Similarly, from bob's perspective, he will randomly sample $y \leftarrow\!\!\$\ Z_m$, then perform the same exponentiation to find $Y = g^y$.

From there, alice and bob will exchange their respective X and Y values. This is the only time Eve will be able to access any information (both X and Y). Now, individually depending on the information they have accessible to them, alice and bob can agree on the same key through

$$K = Y^x = g^{xy} = X^y$$

This process is known as the **Diffie-Hellman Key Exchange**. See below for a diagram for intuition:



Based on this information, there are only a few select ways for Eve to compute K:

- From X, compute $x = \mathsf{DLog}_g(X)$ and then $K = Y^x$
- From Y, compute $y = \mathsf{DLog}_g(Y)$ and then $K = X^y$

And this is why **we need Discrete Log to be computationally hard**. Let's now formalize this "hardness" assumption.

Definition (Discrete Log Assumption)**.** Formally, we define the Discrete Log assumption to be:

Algorithm 30 oracle $\mathsf{DL}[\mathbb{G}, g]$:

```
1: procedure INIT:
2:     X ←$ 𝔾
3:     return X
4:
5: procedure FIN(x):
6:     if X = g^x then
7:         return 1
8:     else
9:         return 0
```

The advantage is defined as

$$\mathsf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(A) = \Pr[A^{\mathsf{DL}[\mathbb{G},g]} \Rightarrow 1]$$

And we say that the Discrete Logarithm (DL) assumption holds in $\mathbb{G}$ with respect to g if $\mathsf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(A)$ is negligible for all polynomial-time A.

Important to note here that running-times / negligibility is typically parameterized by $\log |\mathbb{G}|$.

The discrete log assumption gives us the power to assume the computational hardness of BHKE. Let's now formalize this thought.

Definition (Computational Diffie-Hellman)**.** Formally, we define Computational Diffie-Hellman as:

Algorithm 31 Oracle $\mathsf{CDH}[\mathbb{G}, g]$:

procedure INIT:
    $x, y \leftarrow\$ \mathbb{Z}_{|\mathbb{G}|}$
    return $X = g^x$, $Y = g^y$

procedure FIN(Z):
    if $Z = g^{xy}$ **then**
        return 1
    else
        return 0

The advantage is defined as

$$\mathsf{Adv}^{\text{cdh}}_{\mathbb{G},g}(A) = \Pr[A^{\mathsf{CDH}[\mathbb{G},g]} \Rightarrow 1]$$

And we say that the Computational Diffie-Hellman (CDH) assumption hords in $\mathbb{G}$ with respect to g if $\mathsf{Adv}^{\text{cdh}}_{\mathbb{G},g}(A)$ is negligible for all polynomial-time A.

Again, important to note here that running-times / negligibility is typically parameterized by $\log |\mathbb{G}|$.

But even though it's "computationally hard", can we really say that DHKE is secure? Well,

- Although it's now assumed to be hard to compute $K = g^{xy}$, the assumption doesn't tell us anything about *parts* of the key
- Maybe some scheme only uses half of K, and that half of the key is easy to compute

In general, hard to compute $\neq$ secure. And because of this, the security criteria of DHKE is different from and strong than both DL and CDH assumptions.

Let's establish a new goal using some ideas from symmetric encryption: we want K to be as good as some random K' that's independent of the interaction.

So we need some way of establishing *indistinguishability*. We introduce **Decisional Diffie-Hellman**.

Definition (Decisional Diffie-Hellman)**.** Formally, we define Decisional Diffie-Hellman (DDH) to be:

Algorithm 32 Oracle $\mathsf{DDH}_0[\mathbb{G}, g]$:

procedure INIT:
  $x, y \leftarrow \mathbb{Z}_{|\mathbb{G}|}$
  $X \leftarrow g^x$
  $Y \leftarrow g^y$
  $Z \leftarrow g^{xy}$
  return (X, Y, Z)

Algorithm 33 Oracle $\mathsf{DDH}_1[\mathbb{G}, g]$:

procedure INIT:
  $x, y, z \leftarrow \mathbb{Z}_{|\mathbb{G}|}$
  $X \leftarrow g^x$
  $Y \leftarrow g^y$
  $Z \leftarrow g^z$
  return (X, Y, Z)

The advantage is defined as

$$\mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(D) = |\Pr[D^{\mathsf{DDH}_0[\mathbb{G},g]} \Rightarrow 1] - \Pr[D^{\mathsf{DDH}_1[\mathbb{G},g]} \Rightarrow 1]|$$

And we say that the Decision Diffie-Hellman (DDH) holds in $\mathbb{G}$ with respect to g if $\mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(D)$ is negligible for all polynomial-time D.

Once again, runtime and negligibility here is defined by $\log |\mathbb{G}|$.

> **Intuition.** *This indistinguishability criteria should remind us of IND-CPA, and the oracles / advantage as defined follow a very similar idea! Sending the adversary into one of two worlds (one randomly sampled, one computed), and the adversary should not be able to distinguish which "world" they're in.*

Zooming out a bit, we've now formally defined three separate but related Diffie-Hellman problems, and their relationships with each other can be shown with a chain of implications:

$$\mathsf{DDH} \text{ assumption } \rightarrow \mathsf{CDH} \text{ assumption } \rightarrow \mathsf{DL} \text{ assumption}$$

With this chain of implications in mind, how do we build some group $\mathbb{G}$ for DHKE? Naturally, first thought could be $\mathbb{G} = \mathbb{Z}_N^*$. Recall this lemma

Lemma 17.1. *The group $\mathbb{Z}_N^*$ is cyclic if and only if $N = 2, 4, p^k, 2p^k$, where p is an odd prime number.*

Notice that the group sizes $\varphi(N) = 1, 2, p^{k-1}(p-1), p^{k-1}(p-1)$ are all even! This actually presents us with a problem:

DDH is easy in any $\mathbb{G}$ with even order (e.g. $\mathbb{Z}_P^*$)

Let's discuss some potential solutions:

- **Solution 1:** Take $N = p$ where p is prime. This means we can represent p as $p = 2q+1$ for some other prime q. $\varphi(N) = p - 1 = 2q$ and so $\mathbb{Z}_p^*$ has a subgroup of order q. We can then use this subgroup!
- **Solution 2:** OR instead of doing all that, we could just consider changing the rules of the game and *hash* g^{xy}

That second solution sounds reaaalllll enticing. We now introduce the *Hashed* Diffie-Hellman Key Exchange (HDHKE).

Once again, let $\mathbb{G} = \langle g \rangle$ be a cyclic group with order $|\mathbb{G}| = m$. Now, let $\mathbf{H} : \mathbb{G} \to \{0,1\}^k$ be a hash function (collision resistant).

Alice and bob still compute and exchange X and Y the same way, respectively. But now, to calculate the key K, we have

$$K = \mathbf{H}(Y^x) = \mathbf{H}(g^{xy}) = \mathbf{H}(X^y)$$

Now if CDH holds, given X and Y, we know that eve cannot compute g^{xy}. And the only way for eve to predict K is to compute $\mathbf{H}(g^{xy})$.

This brings us onto another problem. How do we choose $(\mathbb{G}, *)$ such that DL and CDH assumptions are hard?

Once again, let $\mathbb{G} = \langle g \rangle$ be cyclic with order m. We can naively brute-force by performing $O(m)$ group operations. But can we do better?

Let $k = \lceil \sqrt{m} \rceil$. Given $y = g^x$, we know by integer division that there exists q and r such that $x = qk + r$ with $q, r < k$. This implies $g^{qk} = y \cdot g^{-r}$.

Using this intuition, we define a clever algorithm.

Algorithm 34 BabyStepGiantStep(y):

$k \leftarrow \lceil \sqrt{m} \rceil$
for $q = 0$ to $k - 1$ **do**
    $G[q] \to g^{q \cdot k}$
for $r = 0$ to $k - 1$ **do**
    if $\exists q : G[q] = y \cdot g^{-r}$ **then**
        return $x = q \cdot k + r$

Intuition. *The intuition here is that by breaking x down using integer division, we now have two parts to "fine-tune"- either the giant-step, which is the quotient itself, or the baby-step, which is the remainder.*

We first pre-calculate all the giant-steps (which takes time $\sqrt{m}$) and then sift through the baby-steps until there is a collision, in which case we've found the q and r needed to recalculate x using the same integer division.

Remark. This algorithm can be implemented in $O(\sqrt{m}\log m)$ time and memory.

Because of the BSGS algorithm, we want to find a group such that the *best* attack runs in time $\Omega(\sqrt{|\mathbb{G}|}$. If we don't pick a good enough $\mathbb{G}$, it's possible to find a solution using an algorithm faster than BSGS which applies specifically to that case.

- BSGS can be extended to run in time $O(\sqrt{q})$, where q is the largest prime-power that divides $m = |\mathbb{G}|$.
- because of this, we need m to have large prime factors

Now as a final note to end off this very packed lecture, we briefly discuss the case of $\mathbb{Z}_N^*$. As it turns out, even if $m = N - 1$ has large prime factors, $\mathbb{Z}_N^*$ is still "weak".

This is because a **theorem** has shown that there exists an algorithm that solves DLog in $\mathbb{Z}_N^*$ with running time $L_N[1/3, \sqrt[3]{32/9}]$, where L_N is "sub-exponential".

18 Lecture 19: Nov. 7th

Today in lecture we finally debunked the grand question of how exactly to pick a "good" curve for DHKE by introducing elliptic curves. However, I personally feel like this lecture felt a little out of place, and though it's cool to see the math, there wasn't much build up or intuition, and more-so just things being thrown at us to be "memorized", which I am not a big fan of.

18.1 Elliptic curves

We began lecture once again with a quick summary of everything we've covered post midterm. Starting with the idea of a discrete logarithm, the underlying algebraic operation for all that we've discussed, and how that applies to Diffie-Hellman Key Exchange.

To clarify our goal once again, **we want a way to pick** $(\mathbb{G}; *)$ **such that the** DLog **operation is hard**. We know that once we pick such a group, then it will run in time approximately $O(\sqrt{N})$, which is exponential in the size of the group, by the "Baby Step, Giant Step" algorithm.

In general, $\mathbb{Z}_N^*$ is not a great group for cryptography, since DLog is *at best* sub-exponential (assuming BSGS is the best possible). What we want is a *different* approach to build cryptography-friendly groups for which:

- DL, CDH, DDH assumptions are all assumed to be true
- Best known attack is BSGS
- Has the smallest possible element size. For example, a 256-bit element would ideally take complexity 2^{128} to break DLog and CDH.

This is where we introduce **Elliptic Curves**.

Definition (Elliptic Curves)**.** For prime $p > 3$, an **elliptic curve** E over $\mathbb{Z}_p$ is an equation

$$y^2 = x^3 + Ax + B$$

where $A, B \in \mathbb{Z}_p$ such that $4a^3 + 27B^2 \neq 0 \mod p$.

Specifically, the set of points of E is

$$E(\mathbb{Z}_p) = \{\infty\} \cup \{(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p : y^2 = x^3 + Ax + B \mod p\}$$

When representing a curve element, we use $(x, \pm)$

- The coordinate $x^3 + Ax + B$ already defines y^2, and now two roots are both accessible
- $\pm$ tells us which root to take

We now give two certified hashtag facts.

Theorem 18.1 (Hasse's Theorem). *The order of an elliptic cuve over the integers is given by*

$$|E(\mathbb{Z}_p)| = p + t + 1$$

for some t such that $|t| \leq 2\sqrt{p}$.

Corollary 18.1.1 (Schoof's Algorithm). *The quantity $|E(\mathbb{Z}_p)|$ can be computed in time polynomial in $\log p$ (and is efficient even for very large p).*

Let's now discuss how we add these points. We assume $E : y^2 = x^3 + Ax + B, P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{Z}_p)$.

We want to define $P \boxplus Q = R = (x_R, y_R)$. There are now multiple cases:

Case 0: $P \boxplus \infty = \infty \boxplus P = P$

Case 1: $x_P \neq x_Q$. We then have

$$\begin{aligned} s &\leftarrow \frac{y_Q - y_P}{x_Q - x_P} \quad (\text{“slope” of the chord}) \\ x_R &\leftarrow s^2 - x_P - x_Q \\ y_R &\leftarrow s(x_P - x_R) - y_P \end{aligned}$$

Case 2: $P = Q$. Then we have

$$\begin{aligned} s &\leftarrow \frac{3x_p^2 + A}{2y_p} \quad (\text{“slope” of the tangent}) \\ x_R &\leftarrow s^2 - x_P - x_Q \\ y_R &\leftarrow s(x_P - x_R) - y_P \end{aligned}$$

Case 3: $x_P = x_Q$, $x_P = -y_Q$. Then it's simple. $P \boxplus Q = \infty$.

Theorem 18.2. *For any elliptic curve E, is $(E(\mathbb{Z}_p); \boxplus)$ is a group.*

Remark. Not necessarily cyclic! (But often is)

We then ended lecture with a discussion on how exactly elliptic curves are enforced industrially these days, with the 15 standard NIST curves, a common $P256$ curve, and some whacky ones like Curve25519.

Right now, this all may seem very arbitrary, but next lecture when covering public key cryptography everything will hopefully make more sense.

19 Lecture 20: Nov. 10th

This was a hefty lecture! Today we finally dove into the world of public-key encryption, after edging it for so long ever since the midterm. Turns out, key exchange is really just the same thing as a public-key encryption scheme! We discussed two main schemes: El-Gamal (which is just diffie-hellman) and RSA.

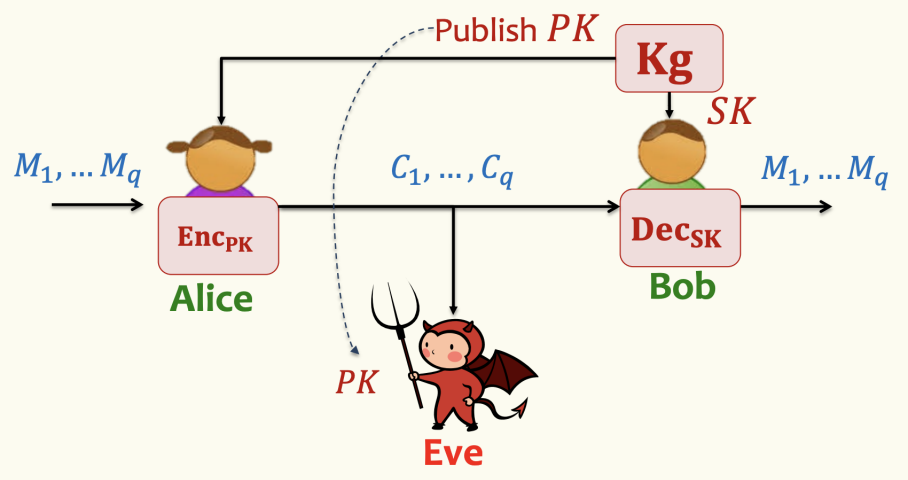
19.1 Public-Key Encryption

To start lecture, we began with a bit of motivation: now that we know how to *exchange* keys through DHKE that was the main point of discussion in the previous few lectures, how do we actually use this?

Well, the low hanging fruit here is just to use what we learned in the first half of the class– with secret key encryption schemes, now that we have a way of sharing keys. This idea is known as **hybrid encryption**.

But there's a different abstraction of this idea as well, a more "standalone" primitive, called **public-key encryption**. Now, rather than the key generation algorithm outputting a secret key, it also outputs a public key that *anyone* can use to encrypt their own messages! But only the person with the secret key will be able to decrypt these messages.

Here's a diagram for intuition:



Definition (Public-key encryption scheme)**.** A **public-key encryption scheme** (often short-handed as PKE) is a triple of algorithms $\Pi = (\textbf{Kg}, \textbf{Enc}, \textbf{Dec})$, where

- The (randomized) **key-generation algorithm Kg** takes no input and output a *public key* PK and a *secret key* SK
- The (randomized) **encryption algorithm Enc** takes as inputs the *public key* PK and the *plaintext* M, and outputs a *ciphertext* $C \leftarrow\!\!\$\ \textbf{Enc}(PK, M)$.

- The (deterministic) **decryption algorithm Dec** is such that $\mathbf{Dec}(SK, \mathbf{Enc}(PK, M)) = M$ for every plaintext M and PK, SK output by $\mathbf{Kg}$.

Here, the message space $\mathcal{M}$ could be $\{0,1\}^n$ or $\{0,1\}^*$.

Now recall that one of the ways we defined "security" with symmetric encryption schemes is using IND-CPA, to require a notion of indistinguishability with respect to different encryptions. This is formalized with an oracle of course.

The same idea should also be carried over to public-key encryption too, right?

Algorithm 35 oracle $\mathsf{LR}_b[\Pi]$

```
procedure INIT:
    (SK, PK) ←$ Kg()
    return PK

procedure ENCRYPT(M^0, M^1):
    if |M^0| ≠ |M^1| then
        return ⊥
    C ←$ Enc(PK, M^b)
    return C
```

And not surprisingly, the ind-cpa advantage is still defined as

$$\mathsf{Adv}^{\text{ind-cpa}}_{\Pi}(D) = |\Pr[D^{\mathsf{LR}_0[\Pi]} \Rightarrow 1] - \Pr[D^{\mathsf{LR}_1[\Pi]} \Rightarrow 1]|$$

However, public-key encryption schemes have a very nice property that is not true for symmetric encryption schemes. This is known as the "one-to-many" property of public-key encryption.

Theorem 19.1 (One-To-Many PKE). *If* Π *is IND-CPA secure for all polynomial-time adversaries making a single Encrypt query, then it is IND-CPA secure for all polynomial time adversaries (making multiple calls to Encrypt).*

> **Intuition.** *This is very unique! It basically means that if some PKE is secure when you encrypt one message, then using the same scheme no matter how many more times, it wall always remain secure*

As a very simple counter-example of why this is *not* the case in symmetric key encryption, consider the one-time pad. It is perfectly secret upon first use, but as we've seen before, when using OTP with the same key multiple times, things quickly begin to fall apart.

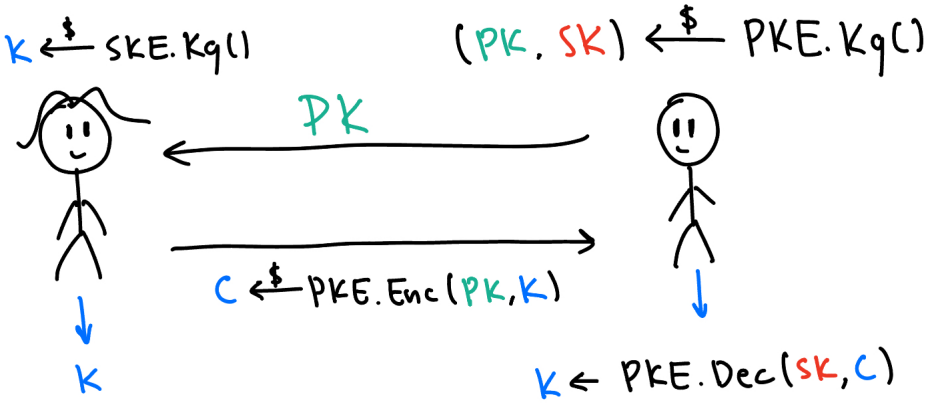
So why, all of a sudden, we're talking about public key encryptions? Well actually, there is a relationship between public-key encryption and the idea of "key exchange" that we've discussed in the previous few lectures. Not to spoil the party, but there actually exists a bijection! Let's discuss this.

Example 19.1. PKE ⇒ Key Exchange.
In other words, the goal here is to take advantage of some public-key encryption scheme to establish a secret key that both parties can use.

Let Bob have access to some PKE and Alice have access to some SKE. To exchange keys without anyone knowing, we follow the steps:

1. Bob generates $(PK, SK) \leftarrow\!\$\ \textbf{PKE.Kg()}$, and Alice generates some $K \leftarrow\!\$\ \textbf{SKE.Kg()}$
2. Bob passes PK to Alice. This is allowed because it's a public key!
3. Alice encrypts the public key $C \leftarrow\!\$\ \textbf{PKE.Enc}(PK, K)$ and passes C to Bob. This is the key step in the "key exchange"
4. Bob obtains $K \leftarrow \textbf{PKE.Dec}(SK, C)$

By IND-CPA security, after seeing PK and C, K still remains pseudo-random.



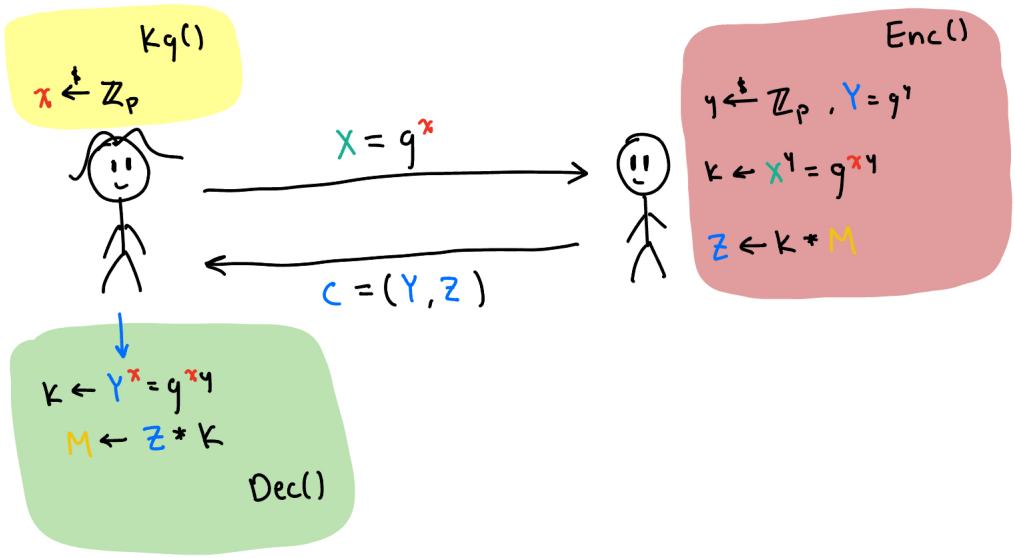
Example 19.2. Key Exchange ⇒ PKE.
This is less obvious, and there are *a lot* of different key exchange protocols out there. The goal here is to "morph" some key exchange protocol into a public-key encryption scheme.

We use DHKE as an example. Here, the public info are the group $\mathbb{G}$, generator g, where $|\mathbb{G}| = p$. We are going to establish a group-theory "One-Time Pad".

1. Alice samples $x \leftarrow\!\$\ \mathbb{Z}_p$. Here, x is analogous to the secret key SK. She then publishes $g^x = X$ as the public key Pk
2. Bob samples $y \leftarrow\!\$\ \mathbb{Z}_p$, and using PK, he obtains some $K = X^y = g^{xy}$.

3. From here, Bob can encrypt some message $M \in \mathbb{G}$ and get the encrypted message $Z = K * M$. He then gives the ciphertext $C = (Z, Y = g^y)$ back to Alice.
4. Alice can decrypt the ciphertext by obtaining $K = Y^x = g^{xy}$, then using it to find $M = Z * K^{-1}$.

Here, step 1 of the process is the key generation, steps 2-3 are the encryption algorithm, and step 4 is the decryption algorithm. We have successfully taken Diffie-Hellman and "transformed" it into a public-key encryption scheme!



Also, one thing to point out here is that we can sort of see why the "one-to-many" PKE property holds here, and that's because it's almost like we're generating a new key every time we call encrypt. It's not entirely the same as a key generation, but since we fix the secret key x, we are generating some new y that acts like a randomly generated key every time.

This example is actually known as the **El-Gamal PKE scheme**. It is IND-CPA secure under the DDH assumption.

Fun story time: Diffie and Hellman, when they invented the revolutionary DHKE, didnt actually realize that they've made a public-key encryption scheme already. So from there, when Rivest, Shamir, and Adleman came up with the RSA public-key encryption scheme, they kinda stole the spotlight and was given the Turing award for creating the "first" public-key encryption scheme (which was revolutionary), even though Diffie and Hellman technically already did it first but just didn't realize it... drama!!!!

19.2 RSA Encryption

The professor began with a blurb about why RSA isn't super desirable anymore and that it is slowly getting ripped out of the curriculum. But for now, because of its historical significance, we are still going to discuss it a bit.

First, to set the picture, we are now interested in $\mathbb{Z}_N^*$ for some $N = P \times Q$ where P and Q are distinct primes. We are now going to heavily rely on Euler's theorem, which states $\forall x \in \mathbb{Z}_N^*$, we have $x^{\varphi(N)} = 1$. In fact, more relevantly, we have the following lemma:

Lemma 19.2. *Suppose $N = PQ$ for distinct primes P, Q, and $e, d \in \mathbb{Z}_{\varphi(N)}^*$ satisfy $ed \bmod \varphi(N) = 1$, then for any $X \in \mathbb{Z}_N$ we have*

$$(X^e)^d \bmod N = X^{ed} \bmod N = X$$

Intuition. *This intuition may not come super... intuitively, but essentially what we're doing when we say X^e is that we're "shuffling" all the elements mod N. Then applying exponentiation by d, we are then "reorganizing" the elements back to their original order. This is what makes RSA work, as we will see soon.*

20 Lecture 21: Nov. 12th

Today we dove into discussing the nitty-gritty details of the RSA encryption, in terms of how it's defined, what makes it "correct", and all the security details that goes along with such a fundamental encryption scheme like this. It was definitely confusing on the first listen, especially with all the little security implications and the build-up up to IND-CCA.

20.1 RSA Encryption (cont'd)

We started off the lecture with a quick recap of what public-key encryption is and what security (ind-cpa) means in the world of PKE. We then reviewed the lemma from Euler's theorem regarding group exponentiation of some X.

From there, we dove in and talked a bit about the plain RSA "encryption". To begin the encryption process, we need to define a couple of variables $PK = (N, e)$, $SK = (N, d)$ where $ed = 1 \pmod{\varphi(N)}$ with $N = PQ$. Now, the encryption and decryptions processes are defined as

$$\text{RSA.Enc}((N, e), M) = M^e \bmod N$$
$$\text{RSA.Dec}((N, d), C) = C^d \bmod N$$

The lemma from the previous lecture should be suffice to show the correctness of this scheme thus far. But is it secure?

Consider this attack: compute $\varphi(N) = (P-1)(Q-1)$, then find some $d \equiv e^{-1} \pmod{\varphi(N)}$. but how do we find $\varphi(N)$ given only N? Turns out, computing $\varphi(N)$ is computationally equivalent to factoring $N = PQ$.

Needless to way, we need a good way to generate some P and Q that makes N "hard" to factor. So how exactly do we generate P and Q?

Algorithm 36 RSA Key Generation

procedure RSA.Kg(k)
  $P \leftarrow\$ \text{SamplePrime}(k)$ // Samples a random k-bit prime
  $Q \leftarrow\$ \text{SamplePrime}(k)$
  $N \leftarrow PQ$
  $e \leftarrow\$ \mathbb{Z}^*_{\varphi(N)}$
  Find d s.t. $ed = 1 \bmod \varphi(N)$
  return $(PK = (N, e), SK = (N, d))$

Given this key generation algorithm, can we conclude that RSA is IND-CPA secure? RSA is **deterministic**, and thus it is not **IND-CPA** secure!

> **Intuition.** *Think about it! If we have the public key, given two plaintexts, we can easily tell which one is encrypted by just encrypting both plaintexts ourselves and checking.*

Alright so, so far our plain RSA has this "flavor of security", but as we can easily see, it's not fool-proof yet. Specifically, we know that given e and d, we can invert the encryption algorithm to decrypt. But we also know that if d is not given, then it will be computationally difficult to find this d, since it will be equivalent to integer factoring.

This idea is known as the RSA assumption: Sample $(N, e, d) \leftarrow\$ \text{RSA.}\mathbf{Kg}(k)$ and $X \leftarrow\$ \mathbb{Z}_N$. Given (N, e) and $X^e \bmod N$ (without d), it is hard to find X.

Algorithm 37 Oracle RSA

```
1: procedure Init
2:     (N, e, d) ←$ RSA.Kg(k)
3:     X ←$ Z_N
4:     Y ← X^e mod N
5:     return (N, e, Y)
6:
7: procedure Fin(X')
8:     if X = X' then
9:         return 1
10:    else
11:        return 0
```

Okay, but how do we get to IND-CPA security? Recall intuitively that IND-CPA stems from the idea of creating randomness between different encryption invocations on the same scheme. To do this, we could use an old trick that we've seen before, but in a different flavor.

Assume we have some $N \geq 2^\lambda$ and plaintext space $\mathcal{M}$, we can use some **randomized padding** procedure $\mathsf{pad} : \mathcal{M} \to \mathbb{Z}_{2^\lambda}$. Similarly, we must also have some unpad procedure to undo the effects.

```
1: procedure Enc(PK = (N, e), M)
2:     X ←$ pad(M)
3:     C ← X^e mod N
4:     return C
```

```
1: procedure Dec(SK = (N, d), C)
2:     X ← C^d mod N
3:     return unpad(X)
```

Let's now discuss one example of a valid padding.

Example 20.1. PKCS#1 Padding

We set the message space to be $\mathcal{M} = \{0,1\}^{\lambda-\rho}$. From here, we define

```
1: procedure pad(M)
2:     R ←$ Z_{2^ρ}
3:     X ← R × 2^{λ−ρ} + M
4:     return X
```

Intuitively, we can think of R as some "randomness" factor, where on line 3, we are bit-shifting this "randomness factor" to the ρ most significant bits.

So to logically unpad this, we have

1: **procedure** UNPAD(X)
2: **if** $X \geq 2^\lambda$ **then**
3: **return** $\perp$
4: **return** $X \bmod 2^{\lambda-\rho}$

This works because we know by division that X is R multiples of $2^{\lambda-\rho}$ plus the message itself, meaning if we simply mod it by $2^{\lambda-\rho}$, we can recover the message.

Fun fact! This is the industry standard to be used with RSA, but it actually hasn't been proved yet if it *truly* is IND-CPA- we just believe it is.

So great, it works now right? End of story? NO.

Take a trip down memory lane... what happened last time when we had something with padding that could output $\perp$? It led to the "padding oracle attack". And that's an issue because a similar model of attack could also be leveraged for our current design, especially so since the adversary can actually encrypt whatever they want to find additional information.

So what do we do here? IND-CPA doesn't seem to be enough of a security requirement. Well, we go again. We introduce a *stronger* notion of security. Our target goal will be some security requirement where *no decryption-related information* can help breaking confidentiality. We call this the **Indistinguishability under Chosen-Ciphertext Attacks** (IND-CCA).

But before we dive into the formality of IND-CCA, pause and think. Why didn't we have something like this in symmetric encryption? Well, because in symmetric encryption we had a *stronger* notion of authenticated encryption- the idea that even if an attack sees a huge amount of encrypted texts, they will not be able to come up with one that gives us a valid decryption.

Why can't we do this with public-key encryption? That's because by design of public-key encryption schemes, we know that **anyone can encrypt anything using the public-key**, meaning regardless of what is being encrypted,the person with the secret key **must be able to decrypt all messages**. This is why we are requiring this new notion of security that has to do with decryption-related information.

Definition (IND-CCA)**.** To formalize, IND-CCA guarantees that adversaries cannot learn anything new from target ciphertext C^* even if they can ask for the decryption of *any* ciphertext $C \neq C^*$ of their choice.

Algorithm 38 oracle $\mathsf{LRD}_\mathsf{b}[\Pi]$

```
1:  procedure INIT
2:      C ← ∅
3:      (SK, PK) ←$ Kg()
4:      return PK
5:
6:  procedure ENCRYPT(M^0, M^1)
7:      if |M^0| ≠ |M^1| then
8:          return ⊥
9:      C ←$ Enc(PK, M^b)
10:     C ← C ∪ {C}
11:     return C
12:
13: procedure DECRYPT(C)
14:     if C ∈ C then
15:         return ⊥
16:     M ← Dec(SK, C)
17:     return M
```

The **ind-cca advantage** of D against Π is

$$\mathsf{Adv}^{\text{ind-cca}}_{\Pi}(D) = |\Pr[D^{\mathsf{LRD}_0[\Pi]} \Rightarrow 1] - \Pr[D^{\mathsf{LRD}_1[\Pi]} \Rightarrow 1]|$$

Intuition. *The intuition behind this CHONKY oracle is that any distinguisher D can make any amount of calls to $\mathsf{LRD}_\mathsf{b}[\Pi]$.decrypt(C), but upon calling $\mathsf{LRD}_\mathsf{b}[\Pi]$.encrypt() with two different plaintexts, D will still not be able to tell which one is encrypted.*

21 Lecture 22: Nov. 14th

Today in lecture we talked about the construction of different RSA attacks when ensuring the different security protocols that we discussed in the previous lecture. From there, we shifted the conversation to talk about "integrity" of public-key encryption schemes. Now, "integrity" here is in quotes because as we've discussed last time, a similar sense of integrity from symmetric encryptions cannot be applied when public-keys are involved. So we instead discussed the idea of a signature.

21.1 RSA Encryption (cont'd)

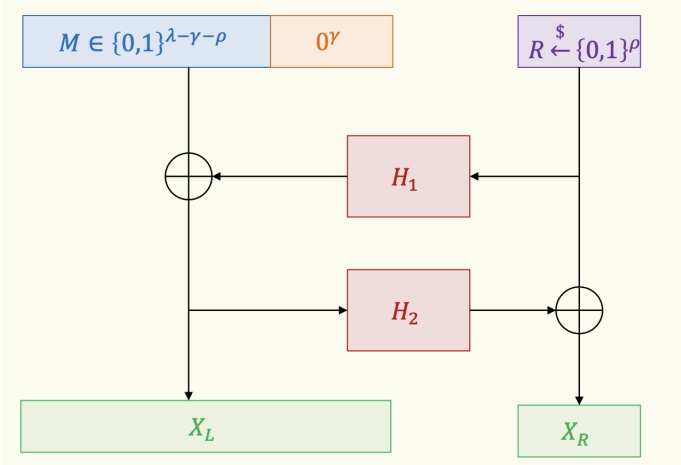
Recall that our goal here with RSA is to come up with a good public-key encryption scheme that relies on "hard math". Specifically, we have some N that is product of two primes P and Q, and based on $\varphi(N)$ (i.e., based on $(P-1)(Q-1)$), we have some encryption exponent that determines how we create our public / secret keys.

Furthermore, recall that in order to make this encryption scheme randomized, we came up with a new PKCS#1 padding scheme that ensures that the encryption satisfies IND-CPA. Note here that the math behind the proof of security is MUCH outside of the scope of this class, so we won't discuss it a bit.

Continuing on with our recollection, we then did a refresher on the new security protocol: IND-CCA, which is a *very powerful* security requirement, as we now also give the adversary the option of decrypting ANY ciphertext they want.

It turns out, with the existence of padding oracle attacks, we found that PKCS#1 RSA encryption is NOT IND-CCA secure. This sets the stage for today's lecture: How can we find a new encryption relating to RSA that satisfies IND-CCA?

We introduced the RSA-OAEP (**O**ptimal **A**symmetric **E**ncryption **P**adding), which is some really wacky padding scheme that is provably strong. We are not going to get into the specifics of *why* in the scope of this course, but do note that it is really quite strong.



Theorem 21.1. *If the RSA assumption is true, RSA-OAEP is IND-CCA secure, assuming that the hash function H_1, H_2 are "ideal".*

Remember that, at the end of the day, this is just the first step of the encryption process. So to decrypt, we would have some ciphertext $C = X^e$, where we take X^{ed} (as per RSA) to recover some $X = X_L \| X_R$. From there, notice that this entire process is reversible, so we simply reverse and recover the original plaintext.

So in summary:

- R, S, and A came up with a hard math problem involving integer factorization that was leveraged to target IND-CPA / IND-CCA security via encryption schemes (the latter is more useful in today's world)
- Proper encryption requires good padding
- Every assumption is based on the RSA assumption (that computing the problem is "hard")

Okay, I'm sick and tired of writing "hard" in quotes. What exactly is hard to break? As discussed before, the RSA assumption is very heavily linked towards the prime factoring problem in math. We now formalize that and say that

$$\text{RSA Assumption} \to \text{Factoring Assumption}$$

The other direction is still an open problem! There's no formal proof yet as to why factoring implies RSA. So how hard is factoring then? Let's naively come up with an algorithm.

Algorithm 39 Factor(N)

```
for i = 2 to ⌈√N⌉ do
    if N mod i = 0 then
        P ← i; Q ← N/P
        return (P, Q)
```

But this is clearly doggy doo-doo! Run time in $O(\sqrt{N}) = L_N[1, 0.5]$ which is a crazy number of iterations for large N! But obviously, this is quite naive. And though there have been other smarter algorithms that run in time "sub-exponential", it hasn't been quite efficient yet.

A very interesting note here though is that every couple of years, we will find new ways of lowering that "sub-exponential" coefficient such that algorithms get a *tiny bit* better, which then means RSA keys would need to get a *tiny bit* longer. This is getting quite frustrating to cryptographers because of the unstable nature of the "hardness of factoring" is.

21.2 Digital Signatures

To start motivating this topic, the prof dropped a really quite cryptic introduction about how this section is going to go: that there will be a clearly defined cryptographic algorithm, but to make sense of it, we would require a lot of human factors(?)

Additionally, so far in our discussion of all of public-key encryption, we've relied on authenticated channels, where the adversary cannot modify what is being sent over that channel.

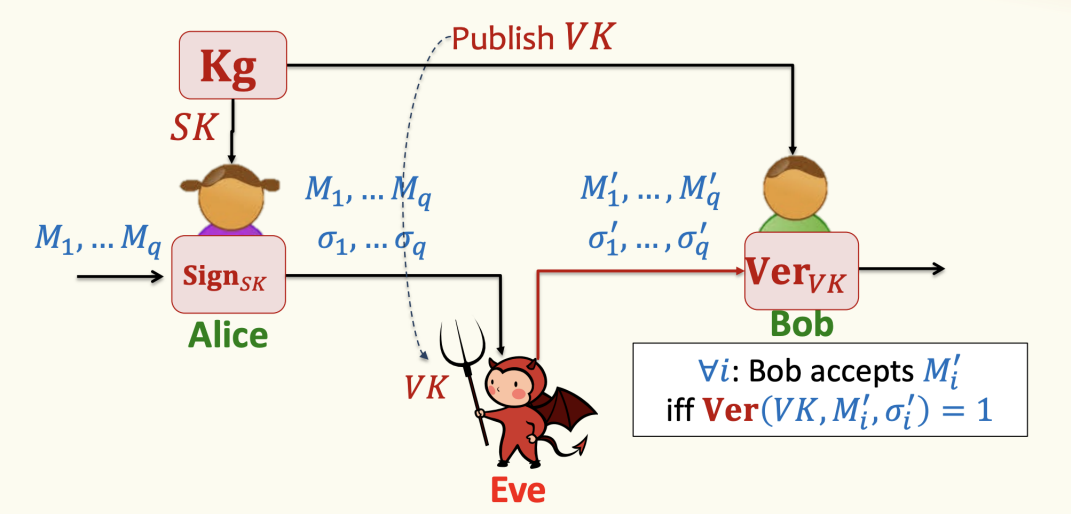
But as with secret-key encryptions, we know that this is not a safe assumption (remember int-ctxt and mac's?). But at the same time, we also talked a bit last lecture about how "authenticated encryption" is meaningless in public-key cryptography because, well, anyone with the public key should be able to encrypt messages anyway, so the whole idea of integrity goes out the window.

OR DOES IT?????

Example 21.1. Back to good ol' Alice n Bob. But rather than putting the responsibility of integrity on Bob, who is being bombarded with a whole bunch of public-key encrypted ciphertexts and has to find out which one is sent by alice, we instead place the responsibility of integrity on alice.

Specifically, we introduce the notion of a public verification key VK and a signing key SK that's secret only to alice, where she can use SK to generate some authentication code $\sigma_1, ..., \sigma_q$ to attach to the end of her message, to which bob only accepts the message after decrypting and verifying with VK that the authentication code is valid.

The intuition for the security criteria now is similar to the notion of *unforgeability under chosen message attack*, where given VK, it would be hard for eve to produce a signature for a new message, even after seeing many many messages.



Let's now formalize this scheme with a definition.

Definition (Digital signature scheme)**.** A **digital signature scheme** is a triple of efficient algorithms $\Sigma = (\textbf{Kg}, \textbf{Sign}, \textbf{Ver})$ where

- **Key generation** algorithm **KG**, takes no inputs and outputs a (random) verification key / signing key pair (VK, SK)
- **Sigining algorithm Sign**, takes input the signing key SK and the plaintext M, outputs *signature* $\sigma \leftarrow \textbf{Sign}(SK, M)$ [Can also be randomized]
- **Verification algorithm Ver** is such that

$$\textbf{Ver}(VK, M, \textbf{Sign}(SK, M)) = 1$$

And as always, the next step in our journey is to formalize the security requirements of this scheme. To do this, let's first recall the **UF-CMA** security requirement for MACs. How can we change that definition to fit our current scheme?

Definition (UF-CMA security for signatures)**.** Formally, we define UF-CMA security for digital signatures as

Algorithm 40 oracle $\mathsf{UF}-\mathsf{CMA}[\Sigma]$

procedure INIT:
  $Q \to \emptyset$
  $(VK, SK) \leftarrow\$ \mathbf{Kg}$
  return VK

procedure EVAL(M):
  $Q \leftarrow Q \cup \{M\}$
  $\sigma \leftarrow \mathbf{Sign}(K, M)$
  return σ

procedure FIN(M^*, σ^*):
  if $M^* \notin Q$ **and** $\mathbf{Ver}(VK, M^*, \sigma^*) = 1$ **then**
    return 1
  else
    return 0

The advantage is defined as

$$\mathsf{Adv}^{\text{uf-cma}}_{\Sigma}(A) = \Pr[A^{\mathsf{UF}-\mathsf{CMA}[\Sigma]} \Rightarrow 1]$$

Σ is (t, ϵ)-$\mathsf{UF}-\mathsf{CMA}$ secure if $\mathsf{Adv}^{\text{uf-cma}}_{\Sigma}(A) \leq \epsilon$ for all A running in time at most t.

Σ is $\mathsf{UF}-\mathsf{CMA}$ secure if $\mathsf{Adv}^{\text{uf-cma}}_{\Sigma}(A)$ is negligible for all polynomial-time A.

Great, now that we've formalized these ideas, how do we actually *build* a good digital signature scheme? Well, why don't we start with considering our good friend RSA. We can treat the "encryption" as the verification key $VK = (N, e)$, and the "decryption" as the signing key $SK = (N, d)$.

Because of RSA hardness, it should imply that forging a signature for some message M *should* be pretty hard right? Does that mean RSA is a good candidate here?

Spoilers: no. it sucks. For one, the signature of the identity 1 will always just be 1. Same thing for the zero element 0. On top of that, if we're given some $\sigma_1 = M_1^d \bmod N$ and $\sigma_2 = M_2^d \bmod N$, we can actually obtain $\sigma_1\sigma_2 = (M_1M_2)^d \bmod N$. This is the same underlying idea behind the padding oracle attack from earlier on, taking advantage of the homomorphic properties of RSA.

22 Lecture 23: Nov. 17th

Today in lecture we wrapped up our discussion of digital signatures, and moved onto the final topic of the course, which is authenticated key exchange, along with the idea of certificates and certification authorities. Lectures like these often get into a lot of the semantics of security in terms of cryptography, rather than strictly math. This is definitely not my strong suit and definitely makes for a lot more definitions and diagrams, which leads to me straight up writing essays in these notes smh.

22.1 Digital Signatures (cont'd)

We began lecture, much as usual, with a brief summary of the topic of focus. Here, we reviewed the definition of a digital signature scheme, and the natural way of defining security with the definition, which is a public-key version of UF-CMA security.

From there, we recalled the traditional way of constructing digital signatures, and that is using plain RSA. We talked about how, in the same way that RSA was not IND-CPA secure as a public-key encryption scheme but was a good starting point, plain RSA signatures are going to be a good starting point as well (despite not being a secure signature).

One a bit more nuanced issue with plain RSA as a signature scheme is that we can only generate signatures for messages in $\mathbb{Z}_N$. But ideally, from a user experience perspective, we would like to generate signatures for any arbitrary length messages.

To solve all of these issues, we introduce a **Full-Domain Hash**. First, consider the hash function $\mathbf{H} : \{0,1\}^s \times \{0,1\}^* \to \mathbb{Z}_N$. We can then define a digital signature scheme as follows:

$VK = (N, e) \quad SK = (N, d) \quad$ with $ed = 1 \pmod{\varphi(N)}$

$$\mathbf{Sign}((N,d), M) = \mathbf{H}(M)^d \bmod N$$
$$\mathbf{Ver}((N,e), M, \sigma) : (\sigma^e \bmod N) == \mathbf{H}(M)?$$

Well, this definitely solves our "arbitrary length" problem, since now we can simply hash our message into an integer in $\mathbb{Z}_N$. But what about security? We have the following theorem:

Theorem 22.1. *Given (1) RSA assumption and (2) the heuristic that* $\boldsymbol{H}$ *acts as a "random" oracle (almost like a PRF), then* ***RSA full-domain-dash is UF-CMA secure****.*

Intuition. *To give a bit of security intuition as to why this is true:*

- *By the random oracle heuristic, output of* $\boldsymbol{H}(M)$ *is random, uniform, and independent*
- *Given* VK*, producing a forged signature based on* (M^*, σ^*) *would require "inverting" the RSA on an uniformly random* $\boldsymbol{H}(M)$*, which is computationally infeasible based on the RSA assumption*
- *Learning extra signatures of* $M \neq M^*$ *does not help, as adversaries can only learn RSA inverse of independent* $\boldsymbol{H}(M)$

So yay! This works! But actually, to quickly shatter our expectations, RSA still sucks in today's world. This is because of the same note that we discussed in the previous lecture, which is the fact that factoring will keep getting *tiny bits* faster every couple years, leading to larger and larger key lengths required for literally *anything* that uses RSA as the underlying algorithm. In today's world, we require RSA signatures to be around 3000+ bits!

So instead, let's talk about something a bit more modern and reliably scalable. To design anything "secure" in the public-key world, we still need a computationally hard underlying math problem. This is where we go back to the *discrete log* problem.

In the mid-1900s, this German guy Schnorr came up with the **Schnorr Signature**, which is a very good digital signature scheme. But then, he decided to patent it, which means ppl have to pay to use it, and obviously no one did, so this genius signature scheme remained unused and not talked about until the patent expired in recent days. Anyway, this digital signature scheme relies on:

Cyclig group $\mathbb{G} = \langle g \rangle$ of prime order p (typically an elliptic curve), and a hash function $\mathbf{H} : \mathbb{G} \times \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p$. The scheme is defined:

- $\mathbf{Kg}()$: $SK \leftarrow\!\!\$\, \mathbb{Z}_p$, $VK \leftarrow g^{SK}$
- $\mathbf{Sign}(SK, M)$:
  - $r \leftarrow\!\!\$\, \mathbb{Z}_p$, $R \leftarrow\!\!\$\, g^r$
  - $z \leftarrow (r + \mathbf{H}(VK, R, M) \times SK) \bmod p$
  - $\sigma = (R, z)$
- $\mathbf{Ver}(VK, M, \sigma = (R, z))$: Check $R \cdot VK^{\mathbf{H}(VK,R,M)} == g^z$

And as for its security, we have the following theorem:

Theorem 22.2. *Given (1) DL assumption and (2) the heuristic that* ***H*** *acts as a "random" oracle (almost like a PRF), then* ***Schnorr Signatures are UF-CMA secure****.*

22.2 Authenticated key exchange

To motivate why we need to talk again about authentication over key exchange, we first considered the assumptions in which we're working over thus far when discussing public-key encryption. We've assumed that key exchanges are done over authenticated channels (that adversaries cannot manipulate our messages), and we've also assumed we *for sure* know who the public-key belongs to.

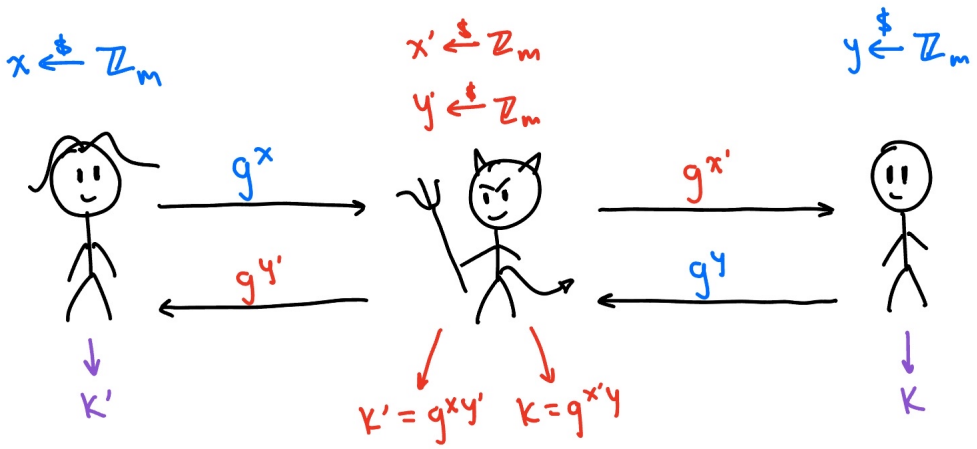
But in reality, neither of these facts are given. For example, if you go on my personal website which supposedly has my personal public-key for you to encrypt a message to me, how do

you know that that's my *actual* public-key? What if someone came along and manipulated my key so that they can read your encrypted messages?

This section is the story behind TLS (HTTPS), SSH, IP-secure, and all that.

To begin, consider Diffie-Hellman Key Exchange that could be done in real life. Recall that the only public information to be observed is the passing of $X = g^x$ and $Y = g^y$ that's being passed to each other by alice and bob. Now suppose, rather than passively watching, eve can actively tamper with the information that's being sent by alice and bob to each other. What can eve do then?

The question here is actually what *can't* eve do? Because once eve is allowed to actively tamper with information, we lose all senses of security what-so-ever. From here, eve can fool alice and bob thinking they're still communicating with each other, while having full control over both the keys as well as all messages being passed through.



Further, remember that key exchanges are really the same underlying thing as public-key encryption schemes, which means public-key encryption schemes are also vulnerable to attacks of a similar idea.

The key idea here is that **"trust" is not a mathematical concept**. We cannot just generate an authenticated channel out of nowhere without some sort of pre-shared information. And this was never an issue before in symmetric encryptions, because we would have already started with some secret key that both parties know, prior to *any* information sharing.

So how do we achieve this in the best way possible in a public-key setting? This is where we introduce the idea of a **public-key infrastructure (PKI)**. Let's discuss a more real-life example than alice and bob and eve

Example 22.1. When we visit google.com, we communicate with the google server via data encrypted using google's public key. The same is done with Facebook, Twitter, and every website in the world. But how do we know we're actually using their key?

We don't. This is actually exactly how some countries' governments control information flow between the domains and their citizens, by replacing the real public-keys of these web servers with the country's self-generated public keys.

So what are some ways around it?

Definition (Public-key infrastructure)**. Public-key infrastructure (PKI)** relies on **certification authority (CA)** that issues **certificates**.

- Certificates link a public key with an entity, and are issued by the CA upon verifying the entity's identity and ownership of a corresponding secret key
- Certificates can be verified with a single verification key PK_{CA} issued by the CA, which needs to be obtained securely.

Remark. This doesn't solve the original problem, it just scales it so that rather than having to trust a ton of independent sources, we can place our trust into one single organization.

23 Lecture 24: Nov. 19th

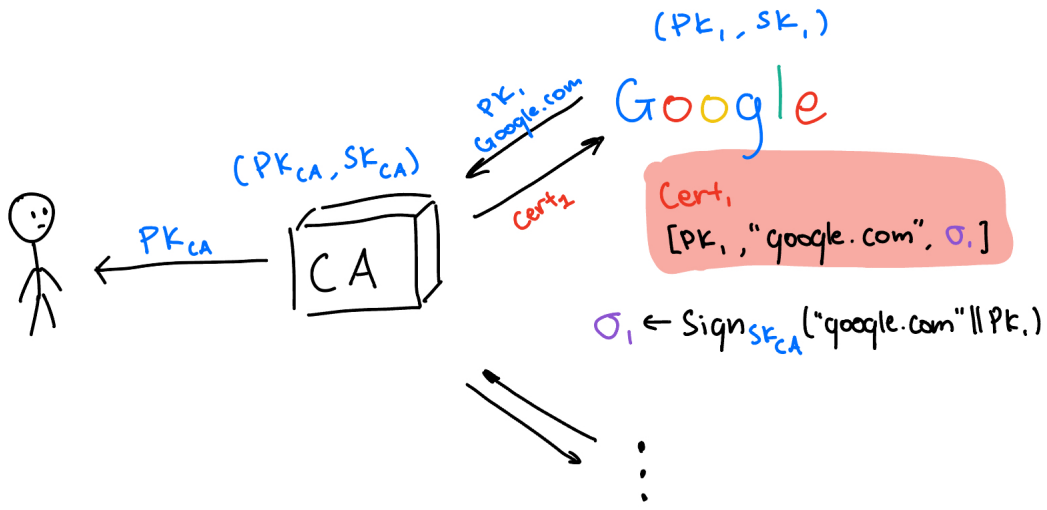
Today in lecture we wrapped up the final ideas on public-key cryptography, giving us a complete picture of exactly how all the puzzle pieces fit together, under the context of TLS and real-world domain visiting. It was actually very cool to see everything we've done within the entire quarter, just abstracted away into a big picture protocol, and just trusting that the little bits and pieces that's been abstracted away function properly.

23.1 Authenticated key exchange (cont'd)

The professor began lecture with a remark that this is going to be the last lecture on traditional cryptography that we will formally cover and that people may encounter in real life.

And with that said, lecture notes beyond this point will probably become more and more intuition-based and lose a bit of rigor, as we approach the end of the quarter.

Today we are going to be discussing the meaning behind the little "padlock" that we see on our browsers– TLS protocol ("HTTPS"). As a recall, we established the idea of authenticated key exchange through the public-key infrastructure system, where there is one "hyper-powerful" organization (the CA) that verifies and certifies the public / secret keys of every domain on the planet.



What we have yet to discuss is how these CAs actually certify domains. As seen with the diagram above, certificates come in the form of $[PK_i, \text{"domain name"}, \sigma_i]$, where $\sigma_i \leftarrow \mathbf{sign}_{SK_{CA}}(\text{"domain name"} \| PK_i)$ is the digital signature that's created by the CA.

So once we have the PK_{CA}, we can use that to verify the certificate of the domain, before using the domain's public key to communicate with it directly. That's the simple picture

at least. But the things we can do with PKI is actually much more nuanced than that.

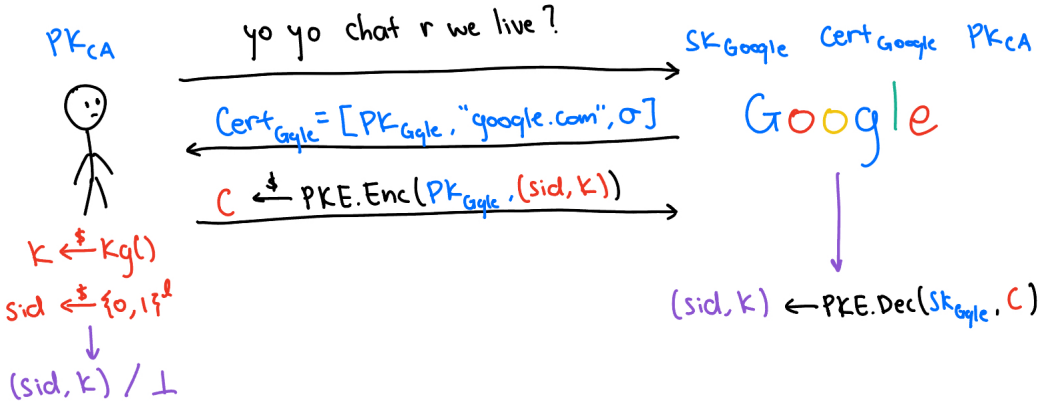
So a typical situation might look like this:

1. Our browser knows and trusts PK_{CA}
2. We type in "google.com"
3. We (our browser) don't trust that our connection is secure

And so from here, if our protocol succeeds,

- we should learn a secret key K
- the only other entity that knows K is "google.com"
- **Google technically still has no idea who we are.** Because of that, we would still need to authenticate via account / password.
  - We *could* also just have a certificate for every single user that exists on the internet – some countries have tried this, but it's generally considered infeasible.

An so how exactly do we agree upon a key with google? Consider the following diagram:

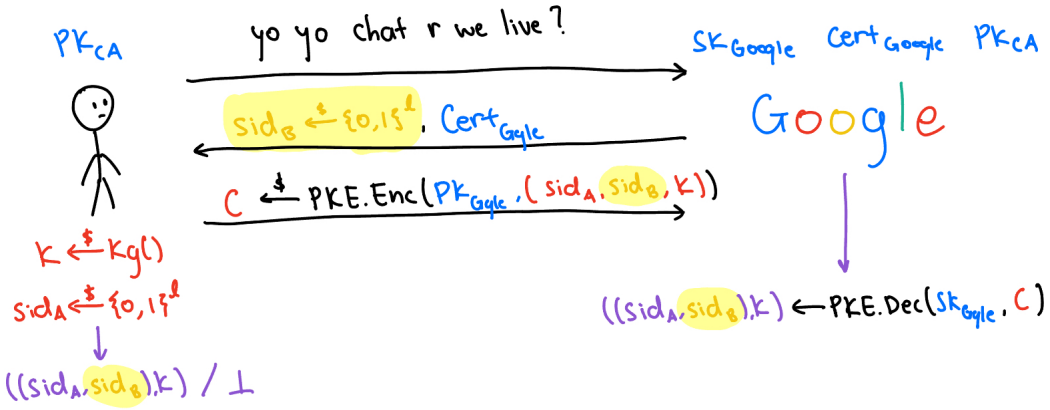


1. client (us) pings the server (Google)
2. server sends its certificate (as defined previously)
3. client confirms that the certificate is valid by checking its signature (as certified by PK_{CA}
4. if valid, client generates a key K and a session id (sid), then encrypts both using the server's public key
5. the server decrypts using the secret key. Now both parties have access to K and sid

Let's now discuss the security that's implied. Consider the potent "adversary-in-the-middle" attack that motivated us to have PKIs in the first place. Because we have a trusted third-party being the CAs, we know that if any adversary tries to tamper the information shared between the client and the server, that tampering *must* be reflected in the certificate, which would lead to suspicion from the client.

So yay! We did it! Is the communication channel secure now? Unfortunately, no, there's always a caveat. It would seem that the current protocol is one-time secure. But consider the scenario at which an adversary simply *observes* this current session (because they still can) and memorize the ciphertext C. Then, at a later time when the current session has closed, they launch a new session by using that same ciphertext C that they memorized previously. Now, they're able to impersonate the client, and the server would never know **because the server doesn't control any of the session-specific information** (which, at the moment, is only K and sid).

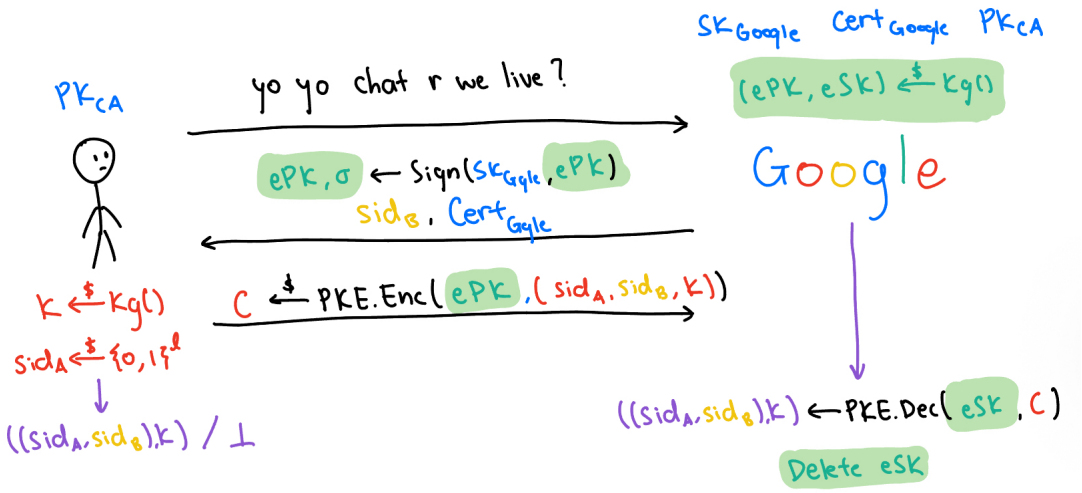
But worry not! The fix is really as simple as it seems: let's let the server also control some session-specific information to prevent this "replay attack".



Now, after decrypting C, the server double checks if sid_B matches and aborts if mismatch occurs.

But actually, there is another slight issue. What if the secret key of the server (SK_{Google} in this case) gets leaked? Obviously, all future communications will now be unreliable, which is to be expected. But with the current implementation, all *past* communications are also vulnerable, as the attacker can now go back to any previous session, and using the leaked SK, can access prior encrypted information.

We need a way to establish what's called **forward-secrecy**. To do this, we require the server to create some temporary, or ephemeral public / secret keys. Basically one-time use keys that's authenticated using the server's permanent keys.



Notice that all encryption / decryption of session-specific information are done with the ephemeral keys, which are immediately deleted subsequently. Further, the client can verify that the ephemeral key they receive actually belongs to the server they're communicating with, because the server is using their permanent SK to sign off these ephemeral keys.

Now, even if an attack had access to the server's secret key, they will not be able to access information from previous sessions. This is the most up-to-date version of essentially what TLS is doing under the hood.

23.2 Hierarchical PKIs

As we've discussed previously, it's infeasible for the entire planet to rely on just a single (or even a few) CAs. We need to produce a *huge* number of certificates, and establishing trust is costly.

This is where the idea of certificate chains come in. CAs actually come in multiple different types and levels, and just accessing a single domain often goes through a chain of 3-4 different CAs.

$(\text{cert}_{\text{root}}, \text{cert}_{\text{CA1}}, \text{cert}_{\text{CA2}}, \text{cert}_{\text{uw.edu}})$

$\text{cert}_{\text{root}} = \text{“root”}||\text{”root”}||PK_{\text{root}}||\sigma$

$\sigma \leftarrow \textbf{Sign}(SK_{\text{root}}, \text{”root”}||\text{”root”}||PK_{\text{root}})$

Self-signed cert!

$\text{cert}_{\text{CA1}} = \text{“CA1”}||\text{”root”}||PK_{\text{CA1}}||\sigma$

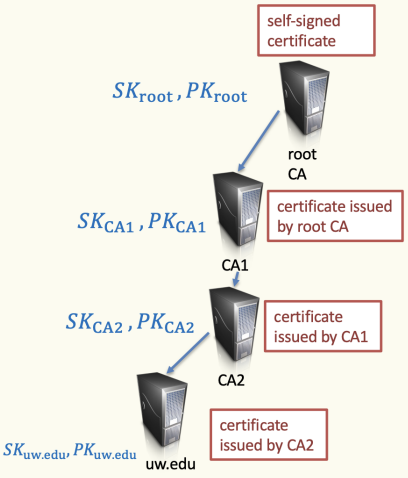
$\sigma \leftarrow \textbf{Sign}(SK_{\text{root}}, \text{”CA1”}||\text{”root”}||PK_{\text{CA1}})$

$\text{cert}_{\text{CA2}} = \text{“CA2”}||\text{”CA1”}||PK_{\text{CA2}}||\sigma$

$\sigma \leftarrow \textbf{Sign}(SK_{\text{CA1}}, \text{”CA2”}||\text{”CA1”}||PK_{\text{CA2}})$

$\text{cert}_{\text{uw.edu}} = \text{“uw. edu”}||\text{”CA2”}||PK_{\text{uw.edu}}||\sigma$

$\sigma \leftarrow \textbf{Sign}(SK_{\text{CA2}}, \text{” uw. edu”}||\text{”CA2”}||PK_{\text{uw.edu}})$



24 Afterwords

Wow... we made it! Well technically there was more than 24 lectures throughout the entire quarter, but some were on non-testable knowledge in which case I didn't bother to jot down super formal notes for. In this last section I think I will just free-write some of the thoughts on this course that I have, and maybe a brief course summary.

Disclaimer: everything beyond this point is personal opinion / interpretation of course content!!

Overall, I think I enjoyed the course very much. In fact, adjacent to taking crypto, I am also taking algebra (ring theory) at the same time, and these two being my only "real" classes these past three months has honestly made this my favorite quarter in college, academically speaking.

I think I found my calling especially during the latter half of the course revoling around computational number theory... shi was so cool fr. Another note, I also really appreciated the chain of reasoning and sorta the "narrative structure" of this course- rather than being a traditional mathematical cryptography course, CSE 426 not only **taught us about cryptographic tools / techniques**, but also about **general concepts allowing for cryptographic thinking**, which are the two main goals that the instructor had outlined.

We began the course in the first week by setting our sights on, at the time, a seemingly very easy goal: what *exactly* does "security" mean? To investigate the answer to this question through a scientific approach, we had to formalize ourselves with some definitions. We started with ideas like the Shannon Secrecy criteria, building into what it means for something to be *perfectly* secret.

Turns out, there are a whole bunch of different ways of extracting information from a cryptographic scheme, and for each of the types of attacks, depending on how much information we're willing to give up to an adversary, "security" can look completely different.

Naturally from there, we dove into the specifics. By introducing the notion of a **symmetric encryption scheme**, we took our first look at a formalized encryption scheme. Much like how everyone and their mothers start with cryptography, we began by looking at the Caesar substitution cipher (and how it sucks).

Using how substitution ciphers suck as motivation, we began introducing different cryptographic objects, such as the block cipher, to help us build "secure" schemes. We also introduced the **Oracle** framework to help us formalize our security, essentially providing ourselves with an interface of what attackers can use, shifting our perspective and allowing us to think like an attacker.

From there, we cruised onwards within the world of symmetric security, establishing security criterias like IND-CPA through formal security proofs, and looking at all different types of adversaries and the advantages that each of them may provide against our scheme.

But then another problem came up: what if instead of worrying about the secrecy of our message, we wanted to also protect our "integrity"? That is, even if an attacker can't READ our messages, we also don't want an attacker to even be able to CREATE a valid message.

And with this new goal in mind, comes a whole ton of new cryptographic objects, definitions, and formalism (which included the likes of hash functions, message authentication codes, INT-CTXT, etc.). And finally, by combining both **secrecy and integrity**, we established the "golden standard" of symmetric cryptography – **Authenticated Encryption**.

And then, the year was 1976, and we were told to take a "*New direction in Cryptography*" (ahahaha im so funny). There's a new problem: everything we've done thus far requires both the sender and the receiver to have some previously established knowledge (a secret key). This is unrealistic in today's world because we need to communicate with people across the internet, and we can't afford the time to meet up at a cafe and share a secret key with everyone we try to talk to.

So how do we exchange keys securely? Turns out this problem required a lot of math, so before we dove into the cryptographic side of things, we first had a 3 lecture long crash course on Group theory (scary math!). **This was my favorite part of the class!!!**

Essentially, key exchange is like a broken puzzle where both the sender and the receiver holds onto a piece. Then they use fancy math, along with the information about the piece that they each have, to trade information securely, allowing the other person to complete the puzzle with the piece that they have.

And this process of trading information is secure because it's guaranteed by math. Unless the attackers have some futuristic super computer, without a piece of the puzzle themselves, they cannot rebuild the puzzle on their own only given the information that's being traded.

This all sounds very abstract, and trust me, it was. Because there's a lot of math out there, we also needed to define what "math" we're using. The two main streams that we learned about were Diffie-Hellman (the good one) and RSA (the famous one).

Also, it turns out that this process of exchanging keys is actually equivalent to encrypting things without a shared secret key, and all of a sudden, we're in the world of **Public-Key Encryption**. And with that, we've opened up a whole different can of worms, which involved a whole new set of definitions and security requirements.

Math allowed us to protect the secrecy of our messages, but a similar problem from a previous part of the class arose again: how do we prevent attackers from CREATING new messages? And this is an inherently harder problem than the one we had to deal with in symmetric encryption, because the fundamental idea of public-key encryption is that *everyone* (yes, even YOU) should be able to encrypt messages that are decrypt-able and readable.

To tackle this problem, we introduced the idea of a signature – if you want your message to be read by the right person, you need to "sign" your message in a way where your signature

can't be forged. This way the receiver will know that the message really came from you, so other attackers can't pretend to be you and send malicious information.

This is now much more akin to real life, where we can encrypt and decrypt messages with people across the internet without a pre-established secret. But there was something else to consider when thinking about real life: **"trust" is not a mathematical concept**, which means we needed something to depend on, and to build clever protocols around it.

And that's exactly how we wrapped up this entire course: with a study on how this type of cryptography worked IRL by going over the semantics and basic protocols that allowed us to securely exchange information with domains on the internet, along with some discussions around the ethics of both sharing and encoding secret information.

Okay that was a lot of free-balling from me, and most of the stuff I just wrote are probably either really oversimplified, or somewhat inaccurate. But yeah, needless to say, I really enjoyed the course!

Below is a (non-exhaustive) list of concepts that were covered:

- Syntax of cryptographic schemes / algorithms
- Security definitions
- Assumptions
- Constructions of schemes
- Security proofs via reductions

And below is a (non-exhaustive) list of cryptographic objects we've interacted with:

- Block ciphers
- Hash function
- Message Authentication Codes (MAC)
- Symmetric (or "secret-key") Encryption Schemes
- Key-exchange Protocol
- Asymmetric (or "public-key") Encryption Schemes
- Signature schemes
- Multi-party computation protocols