M⊕dern Cryptogr🔒phy

or: The Unofficial Notes on the Georgia Institute
of Technology's **CS6260**: *Applied Cryptography*

George Kudrayvtsev
george.k@gatech.edu

Last Updated: January 24, 2020

Preface

I read that Teddy Roosevelt once said, "Do what you can with what you have where you are." Of course, I doubt he was in the tub when he said that.

— Bill Watterson, *Calvin and Hobbes*

Before we begin to dive into all things cryptography, I'll enumerate a few things I do in this notebook to elaborate on concepts:

- An item that is **highlighted like this** is a "term;" this is some vocabulary that will be used and repeated regularly in subsequent sections. I try to cross-reference these any time they come up again to link back to its first defined usage; most mentions are available in the Index.
- An item that is **highlighted like this** is a "mathematical property;" such properties are often used in subsequent sections and their understanding is assumed there.
- An item in a maroon box, like…

  Boxes: **A Rigorous Approach**

  … this often represents fun and interesting asides or examples that pertain to the material being discussed. They are largely optional, but should be interesting to read and have value, even if it's not immediately rewarding.

- An item in a **blue box**, like…

  Quick Maffs: **Proving That the Box Exists**

  … this is a mathematical aside; I only write these if I need to dive deeper into a concept that's mentioned in lecture. This could be proofs, examples, or just a more thorough explanation of something that might've been "assumed knowledge" in the text.

- An item in a green box, like...

Definition 0.1: Example

... this is an important cryptographic definition. It will often be accompanied by a highlighted **term** and dive into it with some mathematical rigor.

I also sometimes include margin notes like the one here (which just links back here) that reference content sources so you can easily explore the concepts further. Linky

INTRODUCTION

Cryptography is hard.

— Anonymous

The purpose of a cryptographic scheme falls into three very distinct categories. A common metaphor used to explain these concepts is a legal document.

- **confidentiality** ensures content *secrecy*—that it can't be read without knowledge of some secret. In our example, this would be like writing the document in a language nobody except you and your recipient understand.
- **authenticity** guarantees content *authorship*—that its author can be irrefutably proven. In our example, this is like signing the original document in pen (assuming, of course, your signature was impossible to forge).
- **integrity** guarantees content *immutability*—that it has not been changed. In our example, this could be that you get an emailed copy of the signed document to ensure that its language cannot be changed post-signing.

Note that even though all three of these properties can go hand-in-hand, they are not mutually constitutive. You can have any of them without the others: you can just get a copy of an unsigned document sent to you in plain English to ensure its integrity later down the line.

Analysing any proposed protocol, handshake, or other cryptographic exchange through the lens of each of these principles will be enlightening. Not every scheme is intended to guarantee all three of them, and different methods are often combined to achieve more than one of these properties. In fact, cases in which only one of the three properties are necessary occur all the time. It's important to not make a cryptographic scheme more complicated than it needs to be to achieve a given purpose: complexity breeds bugs.

TRIVIA: **Cryptography's Common Cast of Characters**

It's really useful to anthropomorphize our discussion of the mathematical intricacies in cryptography. For that, we use a cast of characters whose names give us immediate insight into what we should expect from them.

- **Alice** and **Bob** are the most common sender-recipient pairing. They are generally acting in good faith and aren't trying to break the cryptographic scheme in question. If a third member is necessary, **Carol** will enter the fray (for consistency of the allusion to Lewis Carroll's *Alice in Wonderland* ☺).
- **Eve** and **Mallory** are typically the two members trying to break the scheme. Eve is a *passive* attacker (short for eavesdropper) that merely observes messages between Alice and Bob, whereas malicious Mallory is an *active* attacker who can capture, modify, and inject her own messages into exchanges between other members.

You can check out the Wikipedia article on the topic for more historic trivia and the full cast of characters.

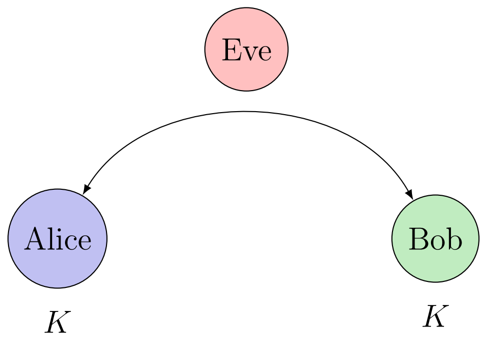
1.1 Overview

There are a number of different categories of cryptography, each of which applies in different settings.

$$\underbrace{\text{crypto}}_{\text{secret}}\underbrace{\text{graphy}}_{\text{writing}}$$

1.1.1 Symmetric Cryptography

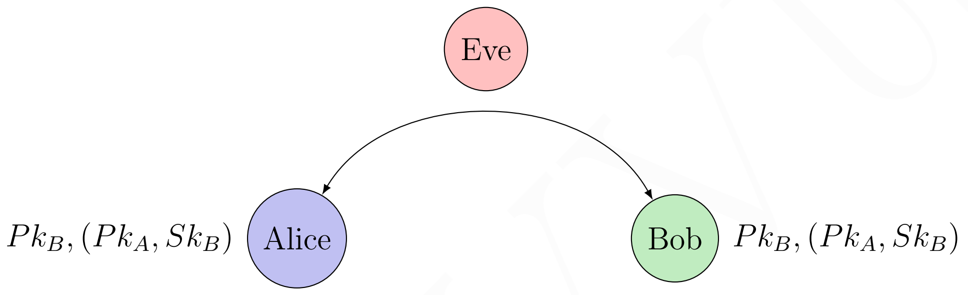
The notion of **symmetric keys** comes from the fact that both the sender and receiver of encrypted information share the same secret key, K. This secret is the only thing that separates a viable receiver from an attacker.



Symmetric key algorithms are often very efficient and supported by hardware, but their fatal flaw lies in **key distribution**. If two parties need to share a secret without anyone else knowing, how do they get it to each other without already having a secure channel?

1.1.2 Asymmetric Cryptography

Asymmetric cryptography is built to solve this problem. Here, secrets are only known to one party; instead, a key (Pk, Sk) is broken into two mathematically-linked components. There is a **public key** that is broadcasted to the world, and a **private key** that must is kept secret.



Asymmetric cryptography is often used to mutually establish a secret key (without revealing it!) for use with symmetric key algorithms, since those are faster. It's a little counterintuitive: two strangers can "meet" and "speak" publicly, yet walk away having established a mutual secret.

The big assumption (which we'll attack later) is that there's a trusted public repository that lists everyone's authentic public keys; without this assumption, we're back at the key distribution problem.

1.1.3 Cryptanalysis

An important part of cryptography is *evaluation*: how good is a given scheme? There are a number of approaches that can help us attack a "secure" protocol:

trial-and-error The most straightforward approach is to identify a possible attack vector and try it out. If it results in a successful attack, the protocol needs to be patched, and you can try finding another vector. If it doesn't seem possible to find an attack vector, that does not make the protocol secure. And yet, how do we find another vulnerability. . . ?

provable security The best way to ensure that a protocol cannot be exploited is to prove it mathematically. It typically relies on proof by contradiction: if an attack is found, then some underlying mathematical assumptions must be false. A good example of this is the factoring problem that underlies most asymmetric cryptography: we've demonstrated time and again the difficulty in factoring large prime numbers. If an efficient algorithm were to arise, a lot of things would go very wrong, very quickly.

Symmetric Cryptography

How long do you want these messages to remain secret? I want them to remain secret for as long as men are capable of evil.

— Neal Stephenson, *Cryptonomicon*

As mentioned in the Introduction, algorithms in symmetric cryptography rely on all members having a shared secret. Let's cover the basic notation we'll be using to describe our schemes and then dive into some.

2.1 Notation & Syntax

For consistency, we'll need common conventions when referring to cryptographic primitives in a scheme.

A well-described symmetric encryption scheme covers the following:

- a **message space**, denoted as the $\mathcal{M}\text{sgSp}$ or $\mathcal{M}$ for short, describes the set of things which can be encrypted. This is typically unrestricted and (especially in the context of the digital world) includes anything that can be encoded as bytes.
- a **key generation algorithm**, $\mathcal{K}$, or the key space spanned by that algorithm, $\mathcal{K}\text{eySp}$, describes the set of possible keys for the scheme and how they are created. The space typically restricts its members to a specific length.
- a **encryption algorithm** and its corresponding **decryption algorithm** $(\mathcal{E}, \mathcal{D})$ describe how a message $m \in \mathcal{M}$ is converted to and from ciphertext. We will use the notation $\mathcal{E}(K, M)$ and $\mathcal{E}_K(M)$ interchangeably to indicate encrypting M using the key K (and similarly for $\mathcal{D}$).

A well-formed scheme *must* allow all valid messages to be en/decrypted. Formally, this means:

$$\forall m \in \mathcal{M}\text{sgSp},\ \forall k \in \mathcal{K}\text{eySp} : \mathcal{D}(k, \mathcal{E}(k, m)) = m$$

An encryption scheme defines the message space and three algorithms: $(\mathcal{M}\text{sgSp}, \mathcal{E}, \mathcal{D}, \mathcal{K})$. The key generation algorithm often just pulls a random n-bit string from the entire $\{0,1\}^n$ bit space; to describe this action, we use notation $K \xleftarrow{\$} \mathcal{K}\text{eySp}$. The encryption algorithm is often randomized (taking random input in addition to (K, M)) and stateful. We'll see deeper examples of all of these shortly.

2.2 One-Time Pads

A **one-time pad** (or OTP) is a very basic and simple way to ensure absolutely perfect encryption, and it hinges on a fundamental binary operator that will be at the heart of many of our symmetric encryption schemes in the future: **exclusive-or**.

Messages are encrypted with an equally-long sequence of random bits using XOR. With our notation, one-time pads can be described as such:

- the key space is all n-bit strings: $\mathcal{K}\text{eySp} = \{0,1\}^n$
- the message space is the same: $\mathcal{M}\text{sgSp} = \{0,1\}^n$
- encryption and decryption are just XOR:

$$\mathcal{E}(K, M) = M \oplus K$$
$$\mathcal{D}(K, C) = C \oplus K$$

Can we be a little more specific with this notion of "perfect encryption"? Intuitively, a secure encryption scheme should reveal nothing to adversaries who have access to the ciphertext. Formally, this notion is called being Shannon-secure (and is also referred to as **perfect security**): the probability of a ciphertext occurring should be equal for any two messages.

Definition 2.1: Shannon-secure

An encryption scheme, $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, is **Shannon-secure** if:

$$\begin{aligned} &\forall m_1, m_2 \in \mathcal{M}\text{sgSp}, \forall C : \\ &\quad \Pr[\mathcal{E}(K, m_1) = C] = \Pr[\mathcal{E}(K, m_2) = C] \end{aligned} \tag{2.1}$$

That is, the probability of a ciphertext C must be equally-likely for any two messages that are run through $\mathcal{E}$.

Shannon security can also be expressed as a conditional probability,[1] where all mes-

[1] See *Definition 2.3* in Katz & Lindell, pp. 29, parts of which are available on Google Books.

sages are equally-probable (i.e. independent of being) given a ciphertext:

$$\forall m \in \mathcal{M}\text{sgSp}, \forall C : \\ \Pr[M = m \,|\, C] = \Pr[M = m]$$

Are one-time pads Shannon-secure under these definitions? Yes, thanks to XOR.

2.2.1 The Beauty of XOR

XOR is the only primitive binary operator that outputs 1s and 0s with the same frequency, and that's what makes it the perfect operation for achieving unpredictable ciphertexts.

Given a single bit, what's the probability of the input bit?

x	y	$x \oplus y$
1	1	0
1	0	1
0	1	1
0	0	0

Table 2.1: The truth table for XOR.

Suppose you have some $c = 0$ (where $c \in \{0, 1\}^1$); what was the input bit m? Well it could've been 1 and been XOR'd with 1 OR it could've been 0 and been XOR'd with 0… Knowing c gives us no new information about the input: our guess is still as good as random chance ($\frac{1}{2} = 50\%$).

Now suppose you know that $c = 1$; are your odds any better? In this case, m could've been 1 and been XOR'd with 0 OR it could've been 0 and XOR'd with 1… Again, we can't do better than random chance.

By the very definition of being Shannon-secure, if we (as the attacker) can't do better than random chance when given a ciphertext, the scheme is perfectly secure.

2.2.2 Proving Security

So we did a bit of a hand-wavy proof to show that XOR is Shannon-secure, but let's be a little more formal in proving that OTPs are perfect as well.

Theorem 2.1. *One-time pads are a perfect-security encryption scheme.*

Proof. We start by fixing an arbitrary n-bit ciphertext: $C \in \{0, 1\}^n$. We also choose a fixed n-bit message, $m \in \mathcal{M}\text{sgSp}$. Then, what's the probability that a randomly-generated key $k \in \mathcal{K}\text{eySp}$ will encrypt that message to be that ciphertext? Namely, what is

$$\Pr[\mathcal{E}(K, m) = C]$$

for our **fixed** m and C? In other words, how many keys can turn m into C?

Well, by the definition of the OTP, we know that this can only be true for a single key: $K = m \oplus C$. Well, since every bit counts, and the probability of a single bit in the key being "right" is $1/2$:

$$\begin{aligned}\Pr[\mathcal{E}(K, m) = C] &= \Pr[K = m \oplus C] \\ &= \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \ldots}_{n \text{ times}} \\ &= \frac{1}{2^n}\end{aligned}$$

Note that this is true $\forall m \in \mathcal{M}\text{sgSp}$, which fulfills the requirement for perfect security! Every message is equally likely to result in a particular ciphertext. □

The problem with OTPs is that keys can only be used once. If we're going to go through the trouble of securely distributing OTPs,[2] we could just exchange the messages themselves at that point in time...

Let's look at what happens when we use the same key across two messages. From the scheme itself, we know that $C_i = K \oplus M_i$. Well if we also have $C_j = K \oplus M_j$, then:

$$\begin{aligned}C_i \oplus C_j &= (K \oplus M_i) \oplus (K \oplus M_j) \\ &= (K \oplus K) \oplus (M_i \oplus M_j) && \text{XOR is associative} \\ &= M_i \oplus M_j && a \oplus a = 0\end{aligned}$$

Though this may seem like insignificant information, it actually can reveal quite a bit about the inputs, and eventually the entire key if it's reused enough times.

An important corrolary of perfect security is what's known as the **impossibility result** (also referred to as the **optimality of the one-time pad** when used in that context):

Theorem 2.2. *If a scheme is Shannon-secure, then the key space cannot be smaller than the message space. That is,*

$$|\mathcal{K}eySp| \geq |\mathcal{M}sgSp|$$

[2] One could envision a literal physical pad in which each page contained a unique bitstring; if two people shared a copy of these pads, they could communicate securely until the bits were exhausted (or someone else found the pad). Of course, if either of them lost track of where they were in the pad, everything would be gibberish from then-on...

Proof. We are given an encryption scheme $\mathcal{E}$ that is supposedly perfectly-secure. So we start by fixing a ciphertext with a specific key, ($K_1 \in \mathcal{K}\text{eySp}$ and plaintext message, $m_1 \in \mathcal{M}\text{sgSp}$):

$$C = \mathcal{E}(K_1, m_1)$$

We know for a fact, then, that at least one key exists that can craft C; thus if we pick a key $K \in \mathcal{K}\text{eySp}$ *at random*, there's a non-zero probability that we'd get C again:

$$\Pr[\mathcal{E}(K, m_1) = C] > 0$$

Suppose then there is a message $m_2 \in \mathcal{M}\text{sgSp}$ which we can *never* get from decrypting C:

$$\Pr[\mathcal{D}(K, C) = m_2] = 0 \qquad \forall K \in \mathcal{K}\text{eySp}$$

By the correctness requirement of a valid encryption scheme, if a message can never be decrypted from a ciphertext, neither should that ciphertext result from an encryption of the message:

$$\Pr[\mathcal{E}(K, M) = C] = 0 \qquad \forall K \in \mathcal{K}\text{eySp}$$

However, that violates Shannon-secrecy, in which the probability of a ciphertext resulting from the encryption of *any* two messages is equal; that's not the case here:

$$\Pr[\mathcal{E}(K, M_1) = C] \neq \Pr[\mathcal{E}(K, M_2) = C]$$

Thus, our assumption is wrong: m_2 cannot exist! Meaning there *must* be some $K_2 \in \mathcal{K}\text{eySp}$ that decrypts C: $\mathcal{D}(K_2, C) = M_2$. Thus, it must be the case that there are as many keys as there are messages. □

Ideally, we'd like to encrypt long messages using short keys, yet this theorem shows that we cannot be perfectly-secure if we do so. Does that indicate the end of this chapter? Thankfully not. If we operate under the assumption that our adversaries are computationally-bounded, it's okay to relax the security requirement and make breaking our encryption schemes very, *very* unlikely. Though we won't have *perfect* secrecy, we can still do extremely well.

We will create cryptographic schemes that are computationally-secure under **Kerckhoff's principle**, which effectively states that *everything* about a scheme should be publicly-available except for the secret key(s).

2.3 Block Ciphers

These are the building blocks of symmetric cryptography: a **block cipher** is a tool for encrypting short strings. Well-known examples include AES and DES.

Definition 2.2: Block Cipher

Formally, a block cipher is a **function family** that maps from a k-bit key and an n-bit input string to an n-bit output string:

$$\mathcal{E} : \{0,1\}^k \times \{0,1\}^n \mapsto \{0,1\}^n$$

Additionally, $\forall K \in \{0,1\}^k$, $\mathcal{E}_K(\cdot)$ is a permutation on $\{0,1\}^n$. This means its inverse is well-defined; we denote it either as $\mathcal{E}_K^{-1}(\cdot)$ or the much more intuitive $\mathcal{D}_K(\cdot)$.

$$\begin{aligned} \forall M, C \in \{0,1\}^n : \qquad & \mathcal{E}_K(\mathcal{D}_K(C)) = C \\ & \mathcal{D}_K(\mathcal{E}_K(M)) = M \end{aligned}$$

In a similar vein, ciphertexts are unique, so $\forall C \in \{0,1\}^n$, there exists a *single* M such that $C = \mathcal{E}_K(M)$.

Math Review: Functions

A function is **one-to-one** if every input value maps to a unique output value. In other words, it's when no two inputs map to the same output.

A function is **onto** if all of the elements in the range have a corresponding input. That is, $\forall y \, \exists x$ such that $f(x) = y$.

A function is **bijective** if it is both one-to-one and onto; it's a **permutation** if it maps a set onto itself. In our case, the set in question will typically be the set of all n-length bitstrings: $\{0,1\}^n$.

2.3.1 Modes of Operation

Block ciphers are limited to encrypting an n-bit string, but we want to be able to encrypt arbitrary-length strings. A **mode of operation** is a way to combine block ciphers to achieve this goal. For simplicitly, we'll assume that our arbitrarily-long messages are actually a multiple of a block length; if they weren't, we could just pad them, but we'll omit that detail for brevity.

ECB—Electronic Code Book

The simplest mode of operation is ECB mode, visually described in Figure 2.1. Given an n-bit block cipher $\mathcal{E}$ and a message of length nb, we could just encrypt it block by block. The decryption is just as easy, applying the inverse block cipher on each piece individually:

$$C[i] = \mathcal{E}_K(M[i])$$
$$M[i] = \mathcal{D}_K(C[i])$$

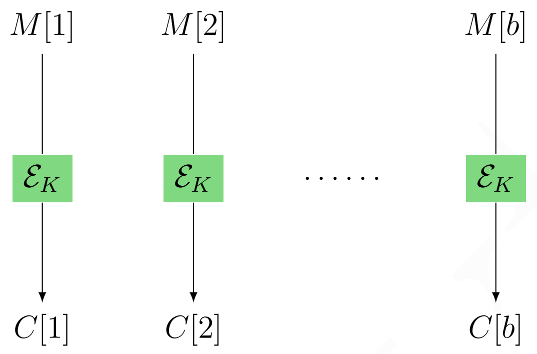


Figure 2.1: The ECB ciphering mode.

This mode of operation has a fatal flaw that greatly compromises its security: if two message blocks are identical, the ciphertexts will be as well. Furthermore, encrypting the same long message will result in the same long ciphertext. This mode of operation is never used, but it's useful to present here to highlight how we'll fix these flaws in later modes.

CBC—Cipher-Block Chaining

This mode of operation fixes both flaws in ECB mode and is usable in real symmetric encryption schemes. It introduces a random **initialization vector** or IV to keep each ciphertext random, and it chains the output of one block into the input of the next block.

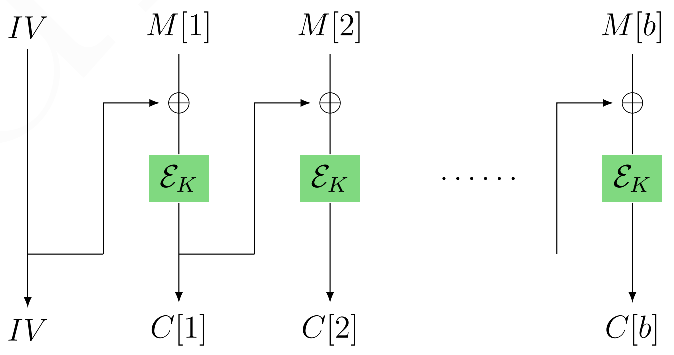


Figure 2.2: The CBC ciphering mode.

Each message block is first chained via XOR with the previous ciphertext before being run through the encryption algorithm. Similarly, the ciphertext is run through the inverse then XOR'd with the previous ciphertext to decrypt. That is,

$$C[i] = \mathcal{E}_K(M[i] \oplus C[i-1])$$
$$M[i] = \mathcal{D}_K(C[i]) \oplus C[i-1]$$

(where the base case is $C[0] = IV$).

The IV can be sent out in the clear, unencrypted, because it doesn't contain any secret information in-and-of itself. If Eve intercepts it, she can't do anything useful with it; if Mallory modifies it, the decrypted plaintext will be gibberish and the recipient will know something is up.

However, if an initialization vector is **repeated**, there can be information leaked to keen attackers about the underlying plaintext.

CBCC—Cipher-Block Chaining with Counter

In this mode, instead of using a randomly-generated IV, a counter is incremented for each new *message* until it wraps around (which typically doesn't occur, consider 2^{128}). This counter is XOR'd with the plaintext to encrypt and decrypt:

$$C[i] = \mathcal{E}_K(M[i] \oplus ctr)$$
$$M[i] = \mathcal{D}_K(C[i]) \oplus ctr$$

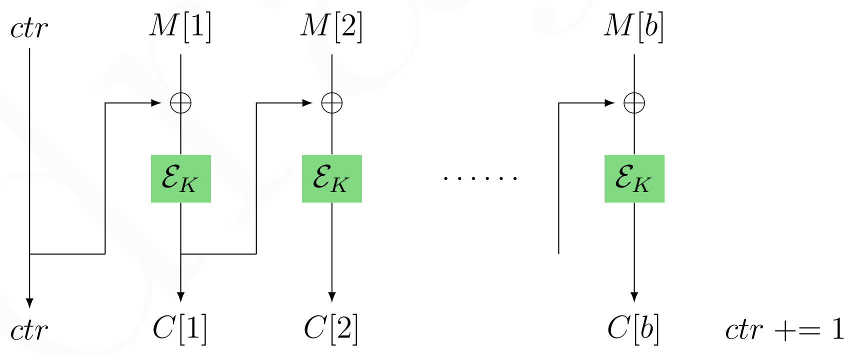


Figure 2.3: The CBCC ciphering mode.

The downside of these two algorithms is not a property of security but rather of performance. Because every block depends on the outcome of the previous block, both encryption and decryption must be done in series. This is in contrast with...

CTR—Randomized Counter Mode

Like ECB mode, this encryption mode encrypts each block independently, meaning it can be parallelized. Unlike all of the modes we've seen so far, though, it does not use a block cipher as its fundamental primitive.[3] Specifically, the encryption function does not need to be invertible. Whereas before we used $\mathcal{E}_K$ as a mapping from a k-bit key and an n-bit string to an n-bit string (see Definition 2.2), we can now use a function that instead maps them to an m-bit string:

$$F : \{0,1\}^k \times \{0,1\}^l \mapsto \{0,1\}^L$$

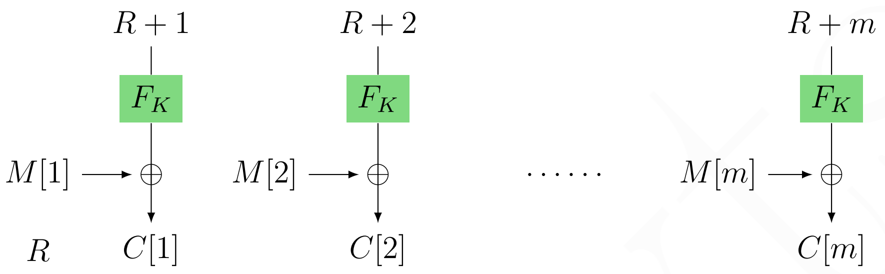


Figure 2.4: The CTR ciphering mode.

This is because both the encryption and decryption schemes use F_K directly. They rely on a randomly-generated value R as fuel, much like the IV in the CBC modes.[4] Notice that to decrypt $C[i]$ in Figure 2.5, one needs to first determine $F_K(R+i)$, then XOR that with the ciphertext to get $M[i]$. The plaintext is never run through the encryption algorithm at all; instead, $F_K(R+i)$ is used as a one-time pad for $M[i]$. That is,

$$C[i] = M[i] \oplus \mathcal{E}_K(R+i)$$
$$M[i] = C[i] \oplus \mathcal{E}_K(R+i)$$

Note that in all of these schemes, the only secret is K (F and $\mathcal{E}$ are likely standardized and known).

CTRC—Stateful Counter Mode

Just like CBC, this mode has a variant that uses a counter rather than a randomly-generated value.

2.4 Evaluating Schemes

Recall that we established that Shannon-secure schemes are impractical, and that we're instead relying on adversaries being computationally bounded to achieve a reasonable level of security. To analyze our portfolio block ciphers, then, we need new definitions of this "computationally-bounded level of security."

[3] In practice, though, F_K will generally be a block cipher. Even though this properly is noteworthy, it does not offer any additional security properties.

[4] In fact, I'm not sure why the lecture decides to use R instead of IV here to maintain consistency. They are mathematically the same: both R and IV are pulled from $\{0,1\}^n$.

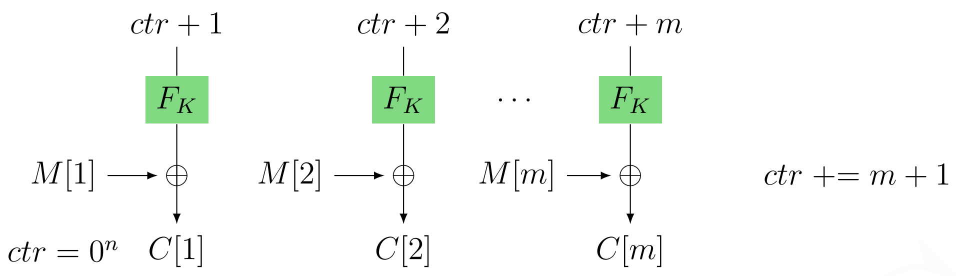


Figure 2.5: The CTRC ciphering mode.

It's easier to reverse the definition: a secure scheme is one that is not insecure. An insecure scheme allows a passive adversary that can see all ciphertexts do malicious things like learn the secret key or read any of the plaintexts. This isn't rigorous enough, though: if the attacker can't see any bits of the plaintext but can compute their sum, is that secure? What if they can tell when identical plaintexts are sent, despite not knowing their content?

There are plenty of possible information leaks to consider and it's impossible to enumerate them all (especially when new attacks are still being discovered!). Dr. Boldyreva informally generalizes the aforementioned ideas:

> *Informally, an encryption scheme is secure if no adversary with "reasonable" resources who sees several ciphertexts can compute any*[5] *partial information about the plaintexts, besides some* a priori *information.*

Though this informality is not useful enough to prove things about encryption schemes we encounter, it's enough to give us intuition on the formal definition ahead.

2.4.1 IND-CPA: Indistinguishability Under Chosen-Plaintext Attacks

It may be a mouthful, but the ability for a scheme to keep all information hidden when an attacker gets to feed their chosen inputs to it is key to a secure encryption scheme. **IND-CPA** is the formal definition of this attack.

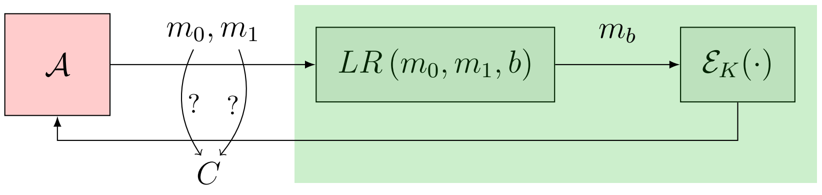
We start with a fixed scheme and secret key, $\mathcal{SE} = (\mathcal{K}\text{eySp}, \mathcal{E}, \mathcal{D})\,; K \xleftarrow{\$} \mathcal{K}\text{eySp}$.

Consider an adversary $\mathcal{A}$ that has access to an oracle. When they provide the oracle with a pair of equal-length messages, $m_0, m_1 \in \mathcal{M}\text{sgSp}$, it outputs a ciphertext.

The oracle, called the "left-right encryption" oracle, chooses a bit $b \in \{0, 1\}^1$ to determine which of these messages to encrypt. It then passes m_b to the encryption function and outputs the ciphertext, $C = \mathcal{E}_K(m_b)$.

The adversary does not know the value of b, and thus does not know which of the

[5] Any information *except* the length of the plaintexts; this knowledge is assumed to be public.



messages was encrypted; it's their goal to figure this out, given full access to the oracle. We say that an encryption scheme is secure if the adversary's ability to determine which subset of ciphertexts corresponds to which experiment is no better than random chance.

Definition 2.3: IND-CPA

A scheme $\mathcal{SE}$ is considered secure under IND-CPA if an adversary's **IND-CPA advantage**—the difference between their probability of guessing correctly and guessing incorrectly—is small (≈ 0):

$$\begin{aligned}\mathsf{Adv}^{\text{ind-cpa}}(\mathcal{A}) = \Pr[&\mathcal{A} \text{ guessed } 0 \text{ for experiment } 0] - \\ \Pr[&\mathcal{A} \text{ guessed } 0 \text{ for experiment } 1]\end{aligned}$$

Since we are dealing with a "computationally-bounded" adversary, $\mathcal{A}$, we need to be cognizant about the real-world meaning behind resource usage. At the very least, we should consider the running time of our scheme and $\mathcal{A}$'s attack. After all, if the encryption function itself takes an entire year, it's likely unreasonable to give the attacker more than a few hundred tries at the oracle before they're time-bound.

We should likewise be cognizant of how many queries the attacker makes and how long they are. We might be willing to make certain compromises of security if, for example, the attacker needs a 2^{512}-length message to gain an advantage.

2.5 Breaking Block Ciphers

With our new formal definition of security under our belt, let's take a crack at breaking the various Modes of Operation we defined. If we can provide an algorithm that demonstrates a reasonable advantage for an adversary that requires reasonable resources, we can show that a scheme is not secure under IND-CPA.

2.5.1 Analysis of ECB

This was clearly the simplest and weakest of schemes that we outlined. The lack of randomness makes gaining an advantage trivial: the message can be determined by having a message with a repeating and one with a non-repeating plaintext.

Algorithm 2.1: A simple algorithm for breaking the ECB block cipher mode.

```
$C_1 \parallel C_2 = \mathcal{E}_K(LR(0^n \parallel 0^n, 0^n \parallel 1^n, b))$
**if** $C_1 = C_2$ **then**
    **return** *0*
**end**
**return** *1*
```

The attack can be generalized to give the adversary perfect knowledge for any input plaintext, and it leads to an important corollary.

Theorem 2.3. *Any deterministic, stateless encryption scheme cannot be IND-CPA secure.*

Proof. Under deterministic encryption, identical plaintexts result in identical ciphertexts. We can always craft a adversary with an advantage. First, we associate ciphertexts with plaintexts, then by the third message we can always determine which ciphertext corresponds to which input.

Algorithm 2.2: A generic algorithm for breaking deterministic encryption schemes.

```
$C_1 = \mathcal{E}_K(LR(0^n, 0^n, b))$
$C_2 = \mathcal{E}_K(LR(1^n, 1^n, b))$
// Given knowledge of these two, we can now always differentiate
   between them.  We can repeat this for any $m \in \mathcal{M}$sgSp.
$C_3 = \mathcal{E}_K(LR(0^n, 1^n, b))$
**if** $C_3 = C_1$ **then**
    **return** *0*
**end**
**return** *1*
```

The proof holds for an arbitrary $\mathcal{M}$sgSp, we chose $\{0,1\}^n$ here for convenience of representation. □

2.5.2 Analysis of CBCC

Turns out, counters are far harder to "get right" relative to random initialization vectors: their predictable nature means we can craft messages that are effectively deterministic by replicating the counter state. Namely, if we pre-XOR our plaintext with the counter, the first ciphertext block functions the same way as in ECB.

The first message lets us identify the counter value. The second message lets us craft a "post-counter" message that will be equal to the third message.

Algorithm 2.3: A simple adversarial algorithm to break CBCC mode.

```
// First, determine the counter (can't count on $ctr = 0$).
$C_0 \parallel C_1 = \mathcal{E}_K(LR(0^n, 1^n, b))$
// Craft a message that'll be all-zeros *post*-counter.
$M_1 = 0^n \oplus (ctr + 1)$
$C_2 \parallel C_3 = \mathcal{E}_K(LR(M_1, 1^n, b))$
// Craft it again, then compare equality.
$M_3 = 0^n \oplus (ctr + 2)$
$C_4 \parallel C_5 = \mathcal{E}_K(LR(M_3, 1^n, b))$
**if** $C_3 = C_5$ **then**
    **return** *0*
**end**
**return** *1*
```

2.5.3 The Rest. . .

It turns out that the other modes of operation are provably IND-CPA secure *if* the underlying blockcipher is secure. Before we dive into those proofs, though, let's define an alternative interpretation of the IND-CPA advantage; we will call this formulation **IND-CPA-cg**, for "chosen guess," and its shown visually in Figure 2.6. This formulation will be more convenient to use in some proofs.

In this version, the choice between left and right message is determined randomly at the start and encoded within b. There is now only one experiment: if the attackers guess matches ($b' = b$), the experiment returns 1.

Definition 2.4: **IND-CPA-cg**

A scheme $\mathcal{SE}$ is still only considered secure under the "chosen guess" variant of IND-CPA if their IND-CPA advantage is small. However, the advantage is now instead defined as:

$$\mathsf{Adv}^{\text{ind-cpa-cg}}(\mathcal{A}) = 2 \cdot \Pr[\text{experiment returns } 1] - 1$$

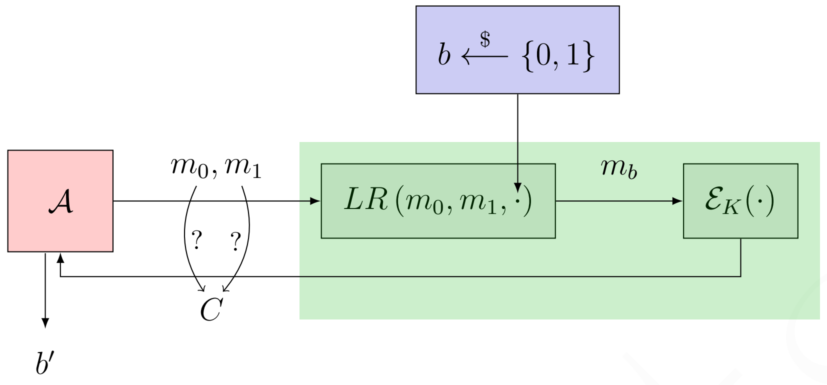


Figure 2.6: The "chosen guess" variant on IND-CPA security, where the attacker must guess a b', and the experiment returns 1 if $b' = b$.

The two variants on attacker advantage in Definition 2.3 and the new Definition 2.4 can be proven equal.

Claim 2.1. $\mathsf{Adv}^{\textit{ind-cpa}}(\mathcal{A}) = \mathsf{Adv}^{\textit{ind-cpa-cg}}(\mathcal{A})$ *for some encryption scheme* $\mathcal{SE}$.

Proof. The probability of the `cg` experiment being 1 (that is, the attacker guessing $b' = b$ correctly) can be expressed as conditional probabilities. Remember that $b \xleftarrow{\$} \{0,1\}$ with uniformally-random probability.

$$\begin{aligned}
\Pr[\text{experiment-cg returns } 1] &= \Pr[b = b'] \\
&= \Pr[b = b' \mid b = 0]\Pr[b = 0] + \Pr[b = b' \mid b = 1]\Pr[b = 1] \\
&= \Pr[b' = 0 \mid b = 0] \cdot \frac{1}{2} + \Pr[b' = 1 \mid b = 1] \cdot \frac{1}{2} \\
&= \frac{1}{2} \cdot \Pr[b' = 0 \mid b = 0] + \frac{1}{2}\left(1 - \Pr[b' = 0 \mid b = 1]\right) \\
&= \frac{1}{2} + \frac{1}{2}\left(\Pr[b' = 0 \mid b = 0] - \Pr[b' = 0 \mid b = 1]\right)
\end{aligned}$$

Notice the expression in parentheses: the difference between the probability of the attacker guessing 0 correctly (that is, when it really is 0) and incorrectly. This is exactly Definition 2.3: advantage under the normal IND-CPA definition! Thus:

$$\begin{aligned}
\Pr[\text{experiment-cg returns } 1] &= \frac{1}{2} + \frac{1}{2}\underbrace{\Pr[b' = 0 \mid b = 0] - \Pr[b' = 0 \mid b = 1]}_{\text{IND-CPA advantage}} \\
&= \frac{1}{2} + \frac{1}{2}\mathsf{Adv}^{\text{ind-cpa}}(\mathcal{A}) \\
\mathsf{Adv}^{\text{ind-cpa-cg}}(\mathcal{A}) &= \mathsf{Adv}^{\text{ind-cpa}}(\mathcal{A})
\end{aligned}$$

$\square$

2.6 Block Ciphers

We can now look into the inner guts of our modes of operation and classify block ciphers as being "secure." Refer to Definition 2.2 to review the mathematical properties of a block cipher. Briefly, it is a function family with a well-defined inverse that maps every message to a unique ciphertext for a specific key.

First off, it's important to recall that we expect attackers to be computationally-bounded to a reasonable degree. This is because block ciphers—and all symmetric encryption schemes, for that matter—are susceptible to an **exhaustive key-search** attack, in which an attacker enumerates every possible $K \in \mathcal{K}\text{eySp}$ until they find the one that encrypts some known message to a known ciphertext. If we say $k = |\mathcal{K}\text{eySp}|$, this obviously takes $O(k)$ time and requires on average 2^{k-1} checks, which is why k must be large enough for this to be infeasible.

Fun Fact: Historical Key Sizes

Modern block ciphers like AES use *at least* 128-bit keys (though 192 and 256-bit options are available) which is considered secure from exhaustive search.

The now-outdated block cipher DES (invented in the 1970s) had a 56-bit key space, and it had a particular property that could speed up exhaustive search by a factor of two. This means exhaustive key-search on DES takes $\approx 2^{54}$ operations which took about 23 years on a 25MHz processor (fast at the time of DES' inception). By 1999, the key could be found in only 22 hours.

The improved triple-DES or 3DES block cipher used 112-bit keys, but it too was abandoned in favor of AES for performance reasons: doing three DES computations proved to be too slow for efficient practical use.

Obviously a block cipher is not necessarily secure just because exhaustive key-search is not feasible. We now aim to define some measure of security for a block cipher. Why can't we just use IND-CPA? Well a block cipher is deterministic *by definition*, and we saw in Theorem 2.3, a deterministic scheme cannot be IND-CPA secure. Thus our definition is too strong! We need something weaker for block ciphers that is still lets us avoid all possible information leaks: nothing about the key, nothing about the plaintexts (or some property of the plaintexts), etc. should be revealed.

We will say that a block cipher is secure if its output ciphertexts "look" random; more precisely, it'd be secure if an attacker can't differentiate its output from a random function. Well... that requires a foray into random functions.

2.6.1 Random Functions

Let's say that $\mathcal{F}(l, L)$ defines the set of ALL functions that map from l-bit strings to L-bit strings:

$$\forall f \in \mathcal{F}(l, L) \qquad f : \{0,1\}^l \mapsto \{0,1\}^L$$

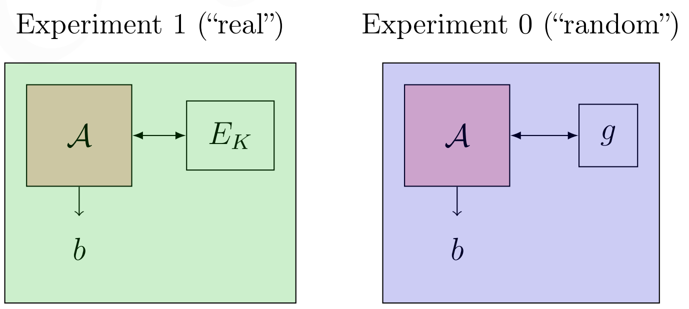
A random function g is then just a random function from that set: $g(\cdot) \xleftarrow{\$} \mathcal{F}(l, L)$. Now because picking a function at random is the same thing as picking a bitstring at random,[6] we can define g in pseudocode as a deterministic way of picking bitstrings:

Algorithm 2.4: $g(x)$, a random function.

```
Define a global array $T$
**if** $T[x]$ *is not defined* **then**
    $T[x] \xleftarrow{\$} \{0,1\}^L$
**end**
**return** $T[x]$
```

A function family is a **pseudorandom function** family (a PRF) if the input-output behavior of a random instance of the family is computationally indistinguishable from a truly-random function. This input-output behavior is defined by algorithm 2.4 and is hidden from the attacker.

2.6.2 Security



The security of a block cipher depends on whether or not an attacker can differentiate between it and a random function. Like with IND-CPA, we have two experiments. In the first experiment, the attacker gets the output of the block cipher E with a fixed $K \in \mathcal{K}$eySp; in the second, it's a random function g chosen from the PRF matching the domain and range of E.

The attacker outputs their guess, b, which should be 1 if they think they're being fed outputs from the real block cipher and 0 if they think it's random. Then, their "advantage" is how much more often the attacker can guess correctly.

[6] Any function we pick will map values to L-bit strings. Concatenating all of these output bitstrings together will result in some nL-bit string, with a $L2^L$ bitstring being longest if the function maps to *every* bitstring. Each chunk of this concatenated string is random, so we can just pick some random $L2^L$-length bitstring right off the bat to pick g.

Definition 2.5: Blockcipher Security

A blockcipher is considered **secure** if an adversary's **PRF advantage** is small (near-zero), where the advantage is defined as the difference in probabilities of the attacker choosing

$$\mathsf{Adv}^{\text{prf}}(\mathcal{A}) = \Pr\left[\mathcal{A} \text{ returns 1 for experiment 1}\right] - \Pr\left[\mathcal{A} \text{ returns 1 for experiment 0}\right]$$

For AES, the PRF advantage is very small and its *conjectured* (not proven) to be PRF secure. Specifically, for running time t and q queries,

$$\mathsf{Adv}^{\text{prf}}(\mathcal{A}) \leq \underbrace{\frac{ct}{T_{\text{AES}}} \cdot 2^{-128}}_{\text{exhaustive key-search}} + \underbrace{q^2 \cdot 2^{-128}}_{\text{birthday paradox}}$$

The second term comes from an interesting attack that can be applied to *all* block ciphers known as the birthday paradox. Recall that block ciphers are permutations, so for distinct messages, you always get distinct ciphertexts. The attack is simple: if you feed the PRF security oracle q distinct messages and get q distinct ciphertexts, you output $b = 1$; otherwise, you output $b = 0$. The only way you get $< q$ distinct ciphertexts is from a g that isn't one-to-one. The probability of this happening is the probability of algorithm 2.4 picking the same bitstring for two xs, so 2^{-L}.

Fun Fact: The Birthday Paradox

Suppose you're at a house party with 50 other people. What're the chances that two people at that party share the same birthday? Turns out, it's really, *really* high: 97%, in fact!

The **birthday paradox** is the counterintuitive idea despite the fact that YOU are unlikely to share a birthday with someone, the chance of ANY two people sharing a birthday is actually extremely high.

In the context of cryptography, this means that as the number of outputs generated by a random function g increases, the probability of SOME two inputs resolving to the same output increases much faster.

Index of Terms