# Applied Cryptography

UT Knoxville, Fall 2023, COSC 583

Scott Ruoti, Alex Zhang

August 26, 2023

# Contents

# 1 Introduction to Applied Cryptography

Applied Cryptography is an in-depth course on cryptography and its application, focusing on modern cryptographic primitives. We will explore the insight behind these primitives, how they relate to each other, and how to apply them to real-world systems. The goals of this course are to:

- Identify commonly used modern cryptographic primitives and how to properly use them.
- Explain how cryptography is used in existing security protocols.
- Write software that leverages the cryptographic primitives covered in this course.

## 1.1 Terminology

When analyzing or designing a security mechanism, we may often ask ourselves, "Is this system secure?" This seems common and a perfectly fine question to ask, but it overlooks the finer details that are essential to security. Instead, security begins by understanding the attackers and threats. Instead, we should be asking ourselves, "Secure from whom?", or "Secure from what?"

### 1.1.1 Actors and Threat

Throughout this course, we will make liberal use of examples to teach new concepts and cement our understanding of them. To facilitate this, we will conform with traditional security literature and follow their naming conventions for actors:

- ***Alice*** and ***Bob*** are the good guys. Many examples involve them communicating between each other, whether it be through email, chat messages, etc.
- ***Eve***, ***Mallory***, and ***Trudy*** are the bad guys. Eve is a passive attacker who will eavesdrop on Alice and Bob. Mallory is an active attacker who will maliciously interfere with Alice and Bob. Trudy will intrude and break into systems.

We will also examine threats, which can be conceptualized in a number of ways. We will often use the ***STRIDE*** threat model developed at Microsoft. It's an acronym that examines:

- spoofing user identity,
- tampering data,
- repudiating (i.e. claiming you never performed an action),
- information disclosure,
- denial of service (DoS), and
- elevation of privileges.

### 1.1.2 Security

It's also important to understand exactly what we mean by a "secure" system. We will examine three facets to security:

1. ***Prevention***: stop bad things from happening in the first place.
2. ***Detection***: if bad things have happened, we want to know about it.
3. ***Reaction***: respond to the bad thing, hopefully resolving any damages and preventing it in the future.

TODO: CIA triad, golden principles

In addition to this, there are two other security objectives we will focus on:

- ***Non-repudiation***: prevent a user from denying they that took a given action, usually involving cryptographic evidence
- ***Deniability***: allow a user to deny that they took a given action (often, this is ineffective in repressive regimes)

Underlying all cryptographic and cybersecurity design, we have the following paradigm:

**Definition 1.1.1 ▶ Kerckhoff's principle**

***Kerckhoff's principle*** states that a cryptosystem should be secure even if everything about the system (except the key) is public knowledge.

In accordance to Kerckhoff's principle, no cryptosystem should ever rely on obscurity for its security. This is almost always the wrong approach, and will almost always be broken. Most modern cryptographic algorithms fully publish the details behind the algorithm, such as AES and RSA.

**Definition 1.1.2 ▶ Weakest link principle**

The ***weakest link principle*** states that a system is only as strong as its weakest link.

**Definition 1.1.3 ▶ Principle of least privilege**

The ***principle of least privilege*** states that processes and users should only be given the privileges needed to perform their expect actions, and nothing more.

### 1.1.3 Threat Modeling

How do we tell what security principles are applicable to our given application? This is a process referred to as threat modeling. The usual workflow for threat modeling is to:

1. Identify the attackers.
2. Decide what threats are relevant.
3. Analyze how the system is impacted by those threats.
4. Identify mitigations for the most impactful threats.

For example, let's consider how Bob might try to send a secure email to Alice. We start by identifying potential attackers, such as rival companies, governments, or even your mom. Next, consider which threats are relevant.

- Eve may try to eavesdrop, breaking confidentiality of the email. This can be mitigated by using an encrypted connection.
- Mallory could perform a man-in-the-middle attack, spoofing herself as the email server to the user, and spoofing herself as the user to the email server. This has the potential to break both confidentiality and integrity. This can be mitigated by using an encrypted connection, endpoint authentication, and digital signatures (which we will cover later).
- Perhaps Trudy might try to simply break into the mail server and download your stored email. This can be mitigated by using end-to-end encryption.

## 1.2 Overview of Cryptography

The goal of cryptography is to conceal data; "cryptography" literally means "hidden writing." Until recently, cryptography has been synonymous with ***symmetric key encryption***. That is, both ciphering a message and deciphering and encrypted message utilized the same secret key.

In this section, we will give a high-level overview of the cryptographic primitives for this course. These include concepts such as the one-time pad, cryptographic hash, symmetric-key cryptography, and public-key cryptography.

**Definition 1.2.1 ▶ Encryption, plaintext, ciphertext, key, cipher**

To ***encrypt*** data is to transform it so that its true meaning is hidden. This requires some form of "special knowledge" to retrieve the original information. In this context, the original message is called the ***plaintext***, and the encrypted message is called the ***ciphertext***. In addition, the "special knowledge" used to decrypt ciphertext is called a ***key***. An encryption algorithm is often called a ***cipher***.

Some examples of classical ciphers include:

- Caesar cipher: shift every letter by some number of letters
- Substitution cipher: assign each letter a new unique letter
- Vignere cipher: a Caesar cipher that can potentially change for each letter

These ciphers all achieve security through secrecy of their inner workings. Generally, all classical ciphers break Kerckhoff's principle. In contrast, modern ciphers adhere to Kerckhoff's principle by simply publishing the details of their inner workings.

Naturally, we may ask: if these modern algorithms are fully known and well-documented, then how can we be sure that they are actually secure? The answer lies in the fact that cryptography often uses ridiculously large numbers.

To put this into perspective, imagine the likelihood that you win the lottery and get struck by lightning all within the same day. There's about a 1 in $2^{61}$ chance of this happening. Often, the smallest key-size still used by cryptographic algorithms today is the 128 bit key, which has $2^{128}$ different permutations. At a much larger scale, there are an estimated $2^{233}$ atoms in the entire Milky Way galaxy. A common key-size to use is 256 bits, which can have $2^{256}$ permutations.

You may ask: couldn't a computer just try each key permutation and eventually find the right one? It turns out that, even with a theoretically perfect computer, it would require the entire energy of the sun for 32 years to cycle all possible 192-bit keys. So it's safe to say that our modern cryptographic algorithms will remain safe for a while.

With this being said, all of this only works when using a sound cryptographic algorithm. The most important takeaway from this entire course is: **never design your own cryptographic algorithms**. It's too easy to get things wrong and too easy to rely on secrecy of the algorithm

itself.

### 1.2.1 Cryptographic Primitives

**Definition 1.2.2 ▶ One-time pad**

The ***one-time pad*** (OTP) is a simple cryptographic algorithm. In essence, we take the bits that represent our message, and perform a bitwise XOR with a secret key to generate ciphertext. To decrypt the ciphertext, we perform the same XOR with the same key.

The one-time pad is one of the few cryptographic algorithms that provides theoretically perfect security. This is because any plaintext can be derived from any ciphertext, depending on the key. How will an adversary know which key to use to decipher to right message? They won't! So why not use one-time pads for all communication? Well...

- The key must be as large as the plaintext.
- A one-time pad key must never be reused. Doing so can lead to much easier guessing and checking by an adversary.
- There are few ways to securely disclose the key to the recipient of the message, especially over the internet.

Many of the modern cryptographic mechanisms still utilize the one-time pad in a more nuanced fashion.

**Definition 1.2.3 ▶ Cryptographic hash, digest**

A ***cryptographic hash*** is a function that takes a variable length input and generates a fixed length output, often called a hash or ***digest***.

Ideally, cryptographic hashes should be:

- one-way operations where the input cannot be determined from the digest alone,
- resistant to collisions (i.e. two inputs are unlikely to have the same digest), and
- very fast to compute.

Cryptographic hashing plays a critical role in many cryptographic algorithms. For examples, it's used in:

- generating message authentication codes,

- generating digital signatures,
- password hashing,
- file integrity verification, and
- rootkit detection.

**Definition 1.2.4 ▸ Symmetric-key cryptography**

***Symmetric-key cryptography*** involves using the same key for both encrypting plaintext to ciphertext and decrypting ciphertext to plaintext.

In symmetric-key cryptography, both the sender and receiver share the same cryptographic key. This provides confidentiality, and with the addition of message authentication codes, it can provide integrity. However, it does not provide non-repudiation in special cases, such as when the key is shared with more than one person, or when a third party examines messages sent using the cryptographic key.

This requires a strong algorithm, resilient to the following kinds of attacks:

**Definition 1.2.5 ▸ Chosen-plaintext attack, chosen-ciphertext attack**

In a ***chosen-plaintext attack***, the attacker can select arbitrary plaintexts and obtain the corresponding ciphertexts. The aim of the attacker is to derive some useful information about the secret key, or the cryptographic mechanism, based on patterns between the plaintexts and ciphertexts.
In a ***chosen-ciphertext attack***, the attacker can select arbitrary ciphertexts and obtain the corresponding plaintexts. The goal here is similar.

Some common symmetric ciphers include:

- Block ciphers like AES, which break plaintext into fixed-size blocks, usually 128 bits per block
- Stream ciphers like ChaCha and RS4, which process plaintext continuously, usually one byte at a time

**Definition 1.2.6 ▸ Message authentication code (MAC)**

A ***message authentication code (MAC)*** is some data that is sent alongside the message. It often uses a symmetric key to lend integrity to the message.

TODO: more about mac?

What can go wrong when using symmetric-key encryption algorithms:

- Relying on the secrecy of the algorithm (e.g. classical ciphers like the Caesar cipher)
- Misusing an algorithm (e.g. WEP using RC4 incorrectly)
- Keys that are too large, which can result in slow operations and may be hard to store
- Keys that are too small, which are vulnerable to brute force attacks

Some symmetric-key use cases include:

- transmitting web pages (e.g. HTTPS),
- encrypted messaging (e.g. email and online chat), and
- cookie integrity.

**Definition 1.2.7 ▶ Public-key cryptography, public key, private key**

***Public-key cryptography*** involves using two keys, one which encrypts plaintext to ciphertext, and another which decrypts ciphertext to plaintext. The key that encrypts is called the ***public key***, while the key that decrypts is called the ***private key***

Public-key cryptography is used in both encryption and digital signatures. The encryption provides confidentiality, and the digital signature provides integrity and non-repudiation. As with symmetric-key cryptography, this requires a strong algorithm that's resilient to chosen-plaintext attacks and chosen-ciphertext attacks.

There are some key distinctions from symmetric-key cryptography. Most notably, it is often significantly slower than symmetric-key cryptography. To combat this, most encrypted messages utilize a technique called ***hybrid encryption***. (TODO: MORE ABOUT HYBRID ENCRYPTION)

## 1.3 Randomness in Cryptography

In accordance to Kerckhoff's principle, a cryptographic mechanism must derive its security from the secrecy of the key alone, not secrecy of the algorithm itself. As such, we need to be sure that when creating a key, it is generated by truly random means. Otherwise, it may be easy to guess, or might cause other issues with the algorithm. The key is not the only randomly generated component; things such as the IV, nonce, and tags must also be random values.

There are generally two types of random number generators. The first is ***true random number generators***.

**Definition 1.3.1 ▶ True random number generator**

A ***true random number generator*** observes randomness from a physical system or process to generate random numbers.

A pitfall of this mechanism is that we may sample from a flawed source of randomness. For example, it may be biased, meaning whether it outputs a 0 or 1 is not a 50/50 split. In addition, it may be correlated, meaning the probability of emitting a 0 or 1 depends on what was previously emitted. Luckily, it's not hard to fix these problems in a process called de-skewing. One possibility is passing bits through a cryptographic hash function.

Common sources of physical randomness include radioactive decay, atmospheric noise, manufacturing differences, and even lava lamps. Common sources of software randomness include the system clock, user inputs, system load, and network statistics. This is often limited by the fact that we need highly-precise measuring equipment to pick up on some of these sources. Moreover, the data itself may be skewed, causing a long and slow de-skewing process.

On the other hand, we have ***pseudo-random number generators***.

**Definition 1.3.2 ▶ Pseudo random number generator**

A ***pseudo random number generator*** applies a mathematical function to the current value to select the next pseudo random value. This results in numbers that appear random (but are not truly random).

One way of making the generated number unguessable is to include secret values in the calculation. Another way is to only return some bits from the calculation.

For the first iteration, we start with some initial value called the seed. For cryptographic uses, we often use a true random number generator to come up with the seed. For non-cryptographic uses, a common choice for seed is the system time, or simply a hard-coded seed by the developer.

An advantage of pseudo-random number generation is that it is typically much faster than true random number generators. If implemented correctly and with the right seed, it will be just as cryptographically secure as a true random number generator.

# Index

## Definitions