5. Introduction to Cryptography

5. Introduction to Cryptography

5.1. Disclaimer: Don't try this at home!

In this class, we will teach you the basic building blocks of cryptography, and in particular, just enough to get a feeling for how they work at a conceptual level. Understanding cryptography at a conceptual level will give you good intuition for how industrial systems use cryptography in practice.

However, cryptography in practice is very tricky to get right. Actual real-world cryptographic implementations require great attention to detail and have hundreds of possible pitfalls. For example, private information might leak out through various side-channels, random number generators might go wrong, and cryptographic primitives might lose all security if you use them the wrong way. We won't have time to teach all of those details and pitfalls to you in CS 161, so you should never implement your own cryptography using the algorithms we teach you in this class.

Instead, the cryptography we show you in this class is as much about educating you as a consumer as educating you as an engineer. If you find yourself needing an encrypted connection between two computers, or if you need to send an encrypted message to another person, you should use existing well-vetted cryptographic tools. However, you will often be faced with the problem of understanding how something is supposed to work. You might also be asked to evaluate the difference between alternatives. For that, you will need to understand the underlying cryptographic engineering involved. Similarly, there are sometimes applications that take advantage of cryptographic primitives in non-cryptographic ways, so it is useful to know the primitives. You never know when you might need a hash, an HMAC, or a block cipher for a non-security task that takes advantage of their randomness properties.

In summary, know that we're going to teach you just enough cryptography to be dangerous, but not enough to implement industrial-strength cryptography in practice.

5.2. Brief History of Cryptography

The word "cryptography" comes from the Latin roots *crypt*, meaning secret, and *graphia*, meaning writing. So cryptography is quite literally the study of how to write secret messages.

Schemes for sending secret messages go back to antiquity. 2,000 years ago, Julius Caesar employed what's today referred to as the "Caesar cypher," which consists of permuting the alphabet by shifting each letter forward by a fixed amount. For example, if Caesar used a shift by 3 then the message "cryptography" would be encoded as "fubswrjudskb". With the development of the telegraph (electronic communication) during the 1800s, the need for encryption in military and diplomatic communications became particularly important. The codes used during this "pen and ink" period were relatively simple since messages had to be decoded by hand. The codes were also not very secure, by modern standards.

The second phase of cryptography, the "mechanical era," was the result of a German project to create a mechanical device for encrypting messages in an unbreakable code. The resulting *Enigma* machine was a remarkable feat of engineering. Even more remarkable was the massive British effort during World War II to break the code. The British success in breaking the Enigma code helped influence the course of the war, shortening it by about a year, according to most experts. There were three important factors in the breaking of the Enigma code. First, the British managed to obtain a replica of a working Enigma machine from Poland, which had cracked a simpler version of the code. Second, the Allies drew upon a great deal of brainpower, first with the Poles, who employed a large contingent of mathematicians to crack the structure, and then from the British, whose project included Alan Turing, one of the founding fathers of computer science. The third factor was the sheer scale of the code-breaking effort. The Germans figured that the Enigma was well-nigh uncrackable, but what they didn't figure on was the unprecedented level of commitment the British poured into breaking it, once codebreakers made enough initial progress to show the potential for success. At its peak, the British codebreaking organization employed over 10,000 people, a level of effort that vastly exceeded anything the Germans had anticipated. They also developed electromechanical systems that could, in parallel, search an incredible number of possible keys until the right one was found.

Modern cryptography is distinguished by its reliance on mathematics and electronic computers. It has its early roots in the work of Claude Shannon following World War II. The analysis of the *one-time pad* (discussed in the next chapter) is due to Shannon. The early 1970s saw the introduction of a standardized cryptosystem, DES, by the National Institute for Standards in Technology (NIST). DES answered the growing need for digital encryption standards in banking and other businesses. The decade starting in the late 1970s then saw an explosion of work on a computational theory of cryptography.

5.3. Definitions

Intuitively, we can see that the Caesar cypher is not secure (try all 26 possible shifts and you'll get the original message back), but how can we prove that it is, in fact, insecure? To formally study cryptography, we will have to define a mathematically rigorous framework that lets us analyze the security of various cryptographic schemes.

The rest of this section defines some important terms that will appear throughout the unit.

5.4. Definitions: Alice, Bob, Eve, and Mallory

The most basic problem in cryptography is one of ensuring the security of communications across an insecure medium. Two recurring members of the cast of characters in cryptography are *Alice* and *Bob*, who wish to communicate securely as though they were in the same room or were provided with a dedicated, untappable line. However, they only have available a telephone line or an Internet connection subject to tapping by an eavesdropping adversary, *Eve*. In some settings, Eve may be replaced by an active adversary *Mallory*, who can tamper with communications in addition to eavesdropping on them.

The goal is to design a scheme for scrambling the messages between Alice and Bob in such a way that Eve has no clue about the contents of their exchange, and Mallory is unable to tamper with the contents of their exchange without being detected. In other words, we wish to simulate the ideal communication channel using only the available insecure channel.

5.5. Definitions: Keys

The most basic building block of any cryptographic system (or *cryptosystem*) is the *key*. The key is a secret value that helps us secure messages. Many cryptographic algorithms and functions require a key as input to lock or unlock some secret value.

There are two main key models in modern cryptography. In the *symmetric key* model, Alice and Bob both know the value of a secret key, and must secure their communications using this shared secret value. In the *asymmetric key* model, each person has a secret key and a corresponding *public key*. You might remember RSA encryption from CS 70, which is an asymmetric-key encryption scheme.

5.6. Definitions: Confidentiality, Integrity, Authenticity

In cryptography, there are three main security properties that we want to achieve.

Confidentiality is the property that prevents adversaries from reading our private data. If a message is confidential, then an attacker does not know its contents. You can think about confidentiality like locking and unlocking a message in a lockbox. Alice uses a key to lock the message in a box and then sends the message (in the locked box) over the insecure channel to Bob. Eve can see the locked box, but cannot access the message inside since she does not have a key to open the box. When Bob receives the box, he is able to unlock it using the key and retrieve the message.

Most cryptographic algorithms that guarantee confidentiality work as follows: Alice uses a key to *encrypt* a message by changing it into a scrambled form that the attacker cannot read. She then sends this encrypted message over the insecure channel to Bob. When Bob receives the encrypted message, he uses the key to *decrypt* the message by changing it back into its original form. We sometimes call the message *plaintext* when it is unencrypted and *ciphertext* when it is encrypted. Even if the attacker can see the encrypted ciphertext, they should not be able to decrypt it back into the corresponding plaintext–only the intended recipient, Bob, should be able to decrypt the message.

Integrity is the property that prevents adversaries from tampering with our private data. If a message has integrity, then an attacker cannot change its contents without being detected.

Authenticity is the property that lets us determine who created a given message. If a message has authenticity, then we can be sure that the message was written by the person who claims to have written it.

You might be thinking that authenticity and integrity seem very closely related, and you would be correct; it makes sense that before you can prove that a message came from a particular person, you first have to prove that the message was not changed. In other words, before you can prove authenticity, you first have to be able to prove integrity. However, these are not identical properties and we will take a look at some edge cases as we delve further into the cryptographic unit.

You can think about cryptographic algorithms that ensure integrity and authenticity as adding a seal on the message that is being sent. Alice uses the key to add a special seal, like a piece of tape on the envelope, on the message. She then sends the sealed message over the unsecure channel. If Mallory tampers with the message, she will break the tape on the envelope, and therefore break the seal. Without the key, Mallory cannot create her own seal. When Bob receives the message, he checks that the seal is untampered before unsealing the envelope and revealing the message.

Most cryptographic algorithms that guarantee integrity and authenticity work as follows: Alice generates a *tag* or a *signature* on a message. She sends the message with the tag to Bob. When Bob receives the message and the tag, he verifies that the tag is valid for the message that was sent. If the attacker modifies the message, the tag should no longer be valid, and Bob's verification will fail. This will let Bob detect if the message has been altered and is no longer the original message from Alice. The attacker should not be able to generate valid tags for their malicious messages.

A related property that we may want our cryptosystem to have is *deniability.* If Alice and Bob communicate securely, Alice might want to publish a message from Bob and show it to a judge, claiming that it came from Bob. If the cryptosystem has deniability, there is no cryptographic proof available to guarantee that Alice's published message came from Bob. For example, consider a case where Alice and Bob use the same key to generate a signature on a message, and Alice publishes a message with a valid signature. Then the judge cannot be sure that the message came from Bob–the signature could have plausibly been created by Alice.

5.7: Overview of schemes

We will look at cryptographic primitives that provide confidentiality, integrity, and authentication in both the symmetric-key and asymmetric-key settings.

	Symmetric-key	Asymmetric-key
Confidentiality	Block ciphers with chaining modes (e.g., AES-CBC)	Public-key encryption(e.g., El Gamal, RSA encryption)
Integrity and authentication	MACs (e.g., AES-CBC-MAC)	Digital signatures (e.g., RSA signatures)

In symmetric-key encryption, Alice uses her secret key to encrypt a message, and Bob uses the same secret key to decrypt the message.

In public-key encryption, Bob generates a matching public key and private key, and shares the public key with Alice (but does not share his private key with anyone). Alice can encrypt her message under Bob's public key, and then Bob will be able to decrypt using his private key. If these schemes are secure, then no one except Alice and Bob should be able to learn anything about the message Alice is sending.

In the symmetric-key setting, *message authentication codes (MACs)* provide integrity and authenticity. Alice uses the shared secret key to generate a MAC on her message, and Bob uses the same secret key to verify the MAC. If the MAC is valid, then Bob can be confident that no attacker modified the message, and the message actually came from Alice.

In the asymmetric-key setting, *public-key signatures* (also known as digital signatures) provide integrity and authenticity. Alice generates a matching public key and private key, and shares the public key with Bob (but does not share her private key with anyone). Alice computes a digital signature of her message using her private key, and appends the signature to her message. When Bob receives the message and its signature, he will be able to use Alice's public key to verify that no one has tampered with or modified the message, and that the message actually came from Alice.

We will also look at several other cryptographic primitives. These primitives don't guarantee confidentiality, integrity, or authenticity by themselves, but they have desirable properties that will help us build secure cryptosystems. These primitives also have some useful applications unrelated to cryptography.

- *Cryptographic hashes* provide a one way digest: They enable someone to condense a long message into a short sequence of what appear to be random bits. Cryptographic hashes are irreversible, so you can't go from the resulting hash back to the original message but you can quickly verify that a message has a given hash.
- Many cryptographic systems and problems need a lot of random bits. To generate these we use a *pseudo random number generator*, a process which takes a small amount of true randomness and stretches it into a long sequence that should be indistinguishable from actual random data.
- *Key exchange* schemes (e.g. Diffie-Hellman key exchange) allow Alice and Bob to use an insecure communication channel to agree on a shared random secret key that is subsequently used for symmetric-key encryption.

5.8. Definitions: Kerckhoff's Principle

Let's now examine the threat model, which in this setting involves answering the question: How powerful are the attackers Eve and Mallory?

To consider this question, recall *Kerckhoff's principle* from the earlier notes about security principles:

> Cryptosystems should remain secure even when the attacker knows all internal details of the system. The key should be the only thing that must be kept secret, and the system should be designed to make it easy to change keys that are leaked (or suspected to be leaked). If your secrets are leaked, it is usually a lot easier to change the key than to replace every instance of the running software. (This principle is closely related to *Shannon's Maxim: Don't rely on security through obscurity.*)

Consistent with Kerckhoff's principle, we will assume that the attacker knows the encryption and decryption algorithms.[1] The only information the attacker is missing is the secret key(s).

[1] The story of the Enigma gives one possible justification for this assumption: given how widely the Enigma was used, it was inevitable that sooner or later the Allies would get their hands on an Enigma machine, and indeed they did.

5.9. Definitions: Threat models

When analyzing the confidentiality of an encryption scheme, there are several possibilities about how much access an eavesdropping attacker Eve has to the insecure channel:

1. Eve has managed to intercept a single encrypted message and wishes to recover the plaintext (the original message). This is known as a *ciphertext-only attack*.
2. Eve has intercepted an encrypted message and also already has some partial information about the plaintext, which helps with deducing the nature of the encryption. This case is a *known plaintext attack*. In this case Eve's knowledge of the plaintext is partial, but often we instead consider complete knowledge of one instance of plaintext.
3. Eve can capture an encrypted message from Alice to Bob and re-send the encrypted message to Bob again. This is known as a *replay attack*. For example, Eve captures the encryption of the message "Hey Bob's Automatic Payment System: pay Eve 100" and sends it repeatedly to Bob so Eve gets paid multiple times. Eve might not know the decryption of the message, but she can still send the encryption repeatedly to carry out the attack.
4. Eve can trick Alice to encrypt arbitrary messages of Eve's choice, for which Eve can then observe the resulting ciphertexts. (This might happen if Eve has access to the encryption system, or can generate external events that will lead Alice to sending predictable messages in response.) At some other point in time, Alice encrypts a message that is unknown to Eve; Eve intercepts the encryption of Alice's message and aims to recover the message given what Eve has observed about previous encryptions. This case is known as a *chosen-plaintext attack*.
5. Eve can trick Bob into decrypting some ciphertexts. Eve would like to use this to learn the decryption of some other ciphertext (different from the ciphertexts Eve tricked Bob into decrypting). This case is known as a *chosen-ciphertext attack*.
6. A combination of the previous two cases: Eve can trick Alice into encrypting some messages of Eve's choosing, and can trick Bob into decrypting some ciphertexts of Eve's choosing. Eve would like to learn the decryption of some other ciphertext that was sent by Alice. (To avoid making this case trivial, Eve is not allowed to trick Bob into decrypting the ciphertext sent by Alice.) This case is known as a *chosen-plaintext/ciphertext attack*, and is the most serious threat model.

Today, we usually insist that our encryption algorithms provide security against chosen-plaintext/ciphertext attacks, both because those attacks are practical in some settings, and because it is in fact feasible to provide good security even against this very powerful attack model.

However, for simplicity, this class will focus primarily on security against chosen-plaintext attacks.