Symmetric-Key Cryptography[1]

1 Overview

Over the next several lectures we'll be studying techniques for securing information and communication in several fundamental ways: *confidentiality* (preventing adversaries from reading our private data), *integrity* (preventing them from altering it), and *authenticity* (determining who created a given document). In a nutshell, cryptography is about communicating securely over insecure communications media.

The ideas we'll examine have significant grounding in mathematics, and in general constitute the most systematic and formal set of approaches to security that we'll cover.

We will start by examining *symmetric-key cryptography*, where both endpoints of a communication share the same key. This is the most classical notion of cryptography. We will study several primitives. *Symmetric-key encryption* provides confidentiality: Alice can encrypt her messages under a key K shared with Bob, and then send the ciphertext to Bob; Bob will be use K to decrypt, but eavesdroppers won't be able to learn anything about the messages Alice is sending. *Message authentication codes (MACs)* provide integrity and authenticity, and act more or less like a keyed checksum. When Alice sends a message M, she can append $F_K(M)$, a "checksum" computed on M using key K; now if Bob has the same key K, he can check that the checksum is valid. MACs are designed so that a man-in-the-middle who doesn't know the key K will be unable to modify the message without invalidating the "checksum".

Then, we will examine *public-key cryptography*. *Public-key encryption* provides confidentiality. Bob generates a matching public key and private key, and shares the public key with his correspondents (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, but no one else will be able to learn anything about the message. *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 her correspondents (but does not share her private key with anyone). Alice compute message a digital signature of her message using her private key, and append 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 in transit.

Then, we will study how to combine these basic building blocks to achieve useful things with cryptography.

[1] Material courtesy of David Wagner

2 Disclaimer

Caution! Here lies dragons. 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. You need to understand them at a conceptual level to get a good feeling for how industrial systems use cryptography in practice.

However, using these cryptographic building blocks securely in a real system requires attention to a lot of intricate details—and we won't have time to teach all of those details and pitfalls to you in CS161. Therefore, we do not recommend that you try to implement your own cryptography using the algorithms we teach you in class.

Later in the class, we'll give you some practical advice on what you *should* do instead. For now, know that we're going to teach you just enough to be dangerous, but not enough to implement industrial-strength cryptography in practice.

3 Brief History of Cryptography

The word "cryptography" comes from the Latin roots *crypt*, meaning secret, and *graphia*, meaning writing. So cryptography is 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 simply 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 developement 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 engineering feat. 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.

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 later in these notes) is due to Shannon. The early 1970s saw the the introduction by NIST (the National Institute for Standards in Technology) of a standardized cryptosystem, *DES*. DES answered the growing need for digital encryption standards in banking and other business. The decade starting in the late 1970s then saw an explosion of work on a computational theory of cryptography.

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. In actual fact they only have available a telephone line or an Internet connection subject to tapping by an eavesdropping adversary, *Eve*. 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 content of their exchange. In other words, we wish to simulate the ideal communication channel using only the available insecure channel.

Encryption focuses on ensuring the *confidentiality* of communications between Alice and Bob. In the *symmetric-key model*, Alice and Bob share a secret key K that is unknown to Eve. Generally K is generated in a random fashion, and for now we don't concern ourselves with how Alice and Bob manage to both have copies of it.

Alice encrypts her message M using the key K, and Bob decrypts the received ciphertext (to recover the original message) using the same key K. The unencrypted message M is often referred to as the *plaintext*, and its encryption as the *ciphertext*.

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

To consider this question, recall (from the earlier notes about principles for secure systems) *Kerkhoff's principle*:

> *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* Don't rely on security through obscurity.*)*

Consistent with Kerkhoff's principle, we will assume that Eve knows the encryption and

decryption algorithms.[2] The only information Eve is missing is the secret key K. There are several possibilities about how much access 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).
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.
3. Eve can trick Alice to encrypt messages $M_1, M_2, \ldots, M_n$ 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 M that is unknown to Eve; Eve intercepts the encryption of M and aims to recover M given what Eve has observed about the encryptions of $M_1, M_2, \ldots, M_n$.
4. Eve can trick Bob into decrypting some ciphertexts $C_1, \ldots, C_n$. Eve would like to use this to learn the decryption of some other ciphertext C (different from $C_1, \ldots, C_n$).
5. A combination of cases 3 and 4: 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 (and in particular did not occur as a result of Eve's trickery).

The first case is known as a *ciphertext-only attack*.

The second 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.

The third case is known as a *chosen-plaintext attack*, and the fourth as a *chosen-ciphertext attack*.

The fifth 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 we *can*, i.e., it is in fact feasible to provide good security even against this very powerful attack model. However, to keep the presentation simple, for the moment these notes will focus primarily on security against chosen-plaintext attack. We will come back to chosen-plaintext/ciphertext attacks later.

[2] 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 Engima machine, and indeed they did.

4 One Time Pad

The one time pad is a simple and idealized encryption scheme that helps illustrate some important concepts. In this scheme, Alice and Bob share an n-bit secret key $K = k_1 \cdots k_n$ where the bits $k_1, \ldots k_n$ are picked uniformly at random (they are the outcomes of independent unbiased coin flips).

Suppose Alice wishes to send the n-bit message $M = m_1 \cdots m_n$.

The desired properties of the encryption scheme are:

1. It should scramble up the message, i.e., map it to a ciphertext $C = c_1 \cdots c_n$.
2. Given knowledge of the secret key K, it should be easy to recover M from C.
3. Eve, who does not know K, should get *no* information about M.

Encryption in the one-time pad is very simple: $c_j = m_j \oplus k_j$, where $\oplus$ is the *XOR* (exclusive-or) of the two bits (0 if the two bits are the same, 1 if they are different).[3]

Decryption is equally simple: $m_j = c_j \oplus k_j$.

To sum up, the one-time pad is described by specifying three procedures:

- Key generation: Alice and Bob pick a shared random key K.
- Encryption algorithm: $C = M \oplus K$.
- Decryption algorithm: $M = C \oplus K$.

Let us now analyze how much information Eve gets about the plaintext M by intercepting the ciphertext C. What is the correct measure of this information gained by Eve? It might be the case that Eve had some partial information about M to begin with. Perhaps she knew that the last bit of M is a 0, or that 90% of the bits of M are 1's, or that M is one of BUY! or SELL but we do not know which. The security property will be: intercepting the ciphertext C should give Eve no additional information about the message M (i.e., she should not learn any new information about M beyond what she already knew before she intercepted C).

[3] Since we'll be making heavy use of *XOR* as we explore various cryptographic schemes, be sure to keep in mind its basic properties:

$x \oplus 0 = x$	0 is the identity
$x \oplus x = 0$	x is its own inverse
$x \oplus y = y \oplus x$	commutative property
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$	associative property

One handy identity that follows from these is:

$$x \oplus y \oplus x = y.$$

Let's prove that the one-time pad meets this security property. Consider the following experiment. Suppose Alice has sent one of two messages M_0 or M_1, and Eve has no idea which was sent. Eve tries to guess which was sent by looking at the ciphertext. We will show that Eve's probability of guessing correctly is $1/2$, which is no different than it would be if she had not intercepted the ciphertext at all.

The proof is very simple: for a fixed choice of plaintext M, every possible value of the ciphertext C can be achieved by an appropriate and unique choice of the shared key K: namely $K = M \oplus C$. Since each such key value K is equally likely, it follows that C is also equally likely to be any n-bit string. Thus Eve sees a uniformly random n bit string no matter what the plaintext message was, and thus gets no information about the plaintext.

Here's another way to see that Eve's probability of guessing successfully is $1/2$. Suppose Eve observes the ciphertext C, and she knows that the message M is either M_0 or M_1, but she does not know which. The probability space here has size 2^{n+1}: it represents the 2^n choices for the n-bit key K, as well as Alice's choice of whether to send M_0 or M_1. All 2^{n+1} choices are equally likely. We can imagine that Alice and Bob randomly and uniformly choose the key K; then Alice randomly chooses a bit $b \in \{0, 1\}$, and Alice sends the encryption of M_b. So, if Eve observes that the ciphertext has some specific value C, what is the conditional probability that $b = 0$ given her observation? It is:

$$\begin{aligned}
\Pr[b = 0 | \text{ciphertext} = C] &= \frac{\Pr[b = 0 \wedge \text{ciphertext} = C]}{\Pr[\text{ciphertext} = C]} \\
&= \frac{\Pr[b = 0 \wedge K = M_0 \oplus C]}{\Pr[\text{ciphertext} = C]} \\
&= \frac{1/2 \cdot 1/2^n}{1/2^n} \\
&= \frac{1}{2}.
\end{aligned}$$

The one time pad has a major drawback. As its name suggests, the shared key cannot be reused to transmit another message M'. If the key K is reused to encrypt two messages M and M', then Eve can take the XOR of the two ciphertexts $C = M \oplus K$ and $C' = M' \oplus K$ to obtain $C \oplus C' = M \oplus M'$. This gives partial information about the two messages. In particular, if Eve happens to learn M, then she can deduce the other message M'. (Actually in this case she can reconstruct the key K, too.)

In practice, even if Eve does not know M or M', often there is enough redundancy in messages that merely knowing $M \oplus M'$ is enough to recover most of M and M'. For instance, the US exploited this weakness to read Soviet communications encrypted with the one-time pad, when US cryptanalysts discovered that Soviet officials in charge of generating random keys for the one-time pad got lazy and started re-using old keys.

Consequently, the one-time pad is not secure if the key is used to encrypt more than one message. Alas, this makes it impractical for almost all practical situations.

5 Block Ciphers

In symmetric encryption schemes, Alice and Bob share a random key and use this single key to repeatedly exchange information securely despite the existence of an eavesdropping adversary, Eve. The block cipher is a fundamental building block in implementing a symmetric encryption scheme.

In a block cipher, Alice and Bob share a k-bit random key K, and use this to encrypt an n-bit message into an n-bit ciphertext. In mathematical notation this can be said as follows. There is an encryption function $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Once we fix the key K, we get a function mapping n bits to n bits: $E_K : \{0,1\}^n \to \{0,1\}^n$ defined by $E_K(M) = E(K, M)$. E_K is required to be a *permutation* on the n-bit strings, in other words, it must be an invertible (bijective) function. The inverse mapping of this permutation is the decryption algorithm D_K. Decryption is the reverse of encryption: $D_K(E_K(M)) = M$.

The Advanced Encryption Standard (AES) is an example of a block cipher. It was designed in 1998 by Joan Daemen and Vincent Rijmen, two researchers from Belgium, in response to a competition organized by NIST.

AES uses a block length of $n = 128$ bits and a key length of $k = 128$ bits (it can also support $k = 192$ or $k = 256$ bit keys). It was designed to be extremely fast in both hardware and software. In terms of security, AES has proved to be an impressively strong algorithm. After all these years, the best practical attack known is *exhaustive key search*, where the attacker systematically tries decrypting some ciphertext using every possible key to see which one gives intelligible plaintext. In the case of AES, exhaustive key search requires 2^{128} computations in the worst case (2^{127} on average). This is a large enough number that even the fastest current supercomputers couldn't possibly mount an exhaustive keysearch attack against AES within the lifetime of our Solar system. Thus AES behaves very differently than the one-time pad. Even given a very large number of plaintext/ciphertext pairs, there appears to be no effective way to decrypt any new ciphertexts.

Crypto-theoreticians formalize these security properties of AES by saying: for a randomly chosen key K, E_K "behaves like" a random permutation on the n-bit strings. Formally, we measure the security of the block cipher by performing the following experiment: the adversary, Eve, is given a box which contains either (I) the encryption function E_K with a random key K, or (II) a permutation π on n bits chosen uniformly at random when the box was created. (The type of box given to Eve is randomly selected, but we don't tell Eve which type of box she has been given.) Eve is allowed T steps in which to play with the box. In each step, Eve can supply an input x to the box and receive a corresponding output y from the box (namely, $y = E_K(x)$ for a type-I box, or $y = \pi(x)$ for a type-II box). After playing with the box, Eve must guess whether the box is type I or type II. The "advantage" of the adversary Eve is $\text{Adv}(\text{Eve}) = 2|p - 1/2|$, where p is the probability that Eve guesses correctly which type of box she was given.

Informally, the advantage of Eve measures how effective she is at distinguishing between the block cipher and a truly random permutation. If Eve guesses blindly, she will be correct with

probability $p = 1/2$, so her advantage will be 0. If Eve can always deduce which type of box she was given (so she is perfect at guessing), she will be correct with probability $p = 1$, so her advantage will be 1. Intuitively, a small advantage means that Eve is doing only a little bit better than chance; a large advantage means that Eve has gained significant information about which type of box she received.

If Eve's advantage is at most ϵ then we say that the block cipher is (T, ϵ)-secure. For AES, the above discussion says that if Eve wants advantage $\epsilon = 1$, she needs $T \geq 2^{128}$ steps. In general there is a tradeoff between T and ϵ, and so we could expect that for a secure algorithm such as AES, we have $T/\epsilon \geq 2^{128}$ for all successful attacks. In some sense $\log(T/\epsilon)$ is the "effective key length" of the block cipher, i.e., a measure of how much work Eve must exert as equated with brute-forcing a key of the effective length. For instance, if Eve only has enough budget to do 2^{64} steps of computation, then as far as we know, she can only get advantage of about $1/2^{64}$ at distinguishing AES from a random permutation. This is remarkable: it means that Eve is learning almost nothing.

6 Symmetric Encryption Schemes

A symmetric encryption scheme allows Alice and Bob to privately exchange a sequence of messages in the presence of an eavesdropper Eve. We will assume that Alice and Bob share a random secret key K. How Alice and Bob managed to share a key without the adversary's knowledge is not going to be our concern here (we will talk about it later, though). The encryption scheme consists of an encryption algorithm $\mathcal{E}$ that takes as input the key K and the plaintext message $M \in \{0,1\}^*$, and outputs the ciphertext. The decryption algorithm $\mathcal{D}$ takes as input the key and the ciphertext and reconstructs the plaintext message M.

In general the encryption algorithm builds upon a block cipher to accomplish two goals. The first goal is that we'd like to encrypt arbitrarily long messages using a fixed-length block cipher. The other is to make sure that if the same message is sent twice, the ciphertext in the two transmissions is not the same. To achieve these goals, the encryption algorithm can either be randomized or stateful—it either flips coins during its execution, or its operation depends upon some state information. The decryption algorithm is neither randomized nor stateful.

There are four standard ways of building an encryption algorithm, using a block cipher:

ECB Mode (Electronic Code Book): In this mode the plaintext M is simply broken into n-bit blocks $M_1 \cdots M_l$, and each block is encoded using the block cipher: $C_i = E_K(M_i)$. The ciphertext is just a concatenation of these individual blocks: $C = C_1 \cdot C_2 \cdots C_l$. This scheme is **flawed**. Any redundancy in the blocks will show through and allow the eavesdropper to deduce information about the plaintext. For instance, if $M_i = M_j$, then we will have $C_i = C_j$, which is visible to the eavesdropper; so ECB mode **leaks information** about the plaintext.

CBC Mode (Cipher Block Chaining): This is a popular mode for commercial applications. For each message the sender picks a random n-bit string, called the initial vector or IV.

Define $C_0 = IV$. The i^{th} ciphertext block is given by $C_i = E_K(C_{i-1} \oplus M_i)$. The ciphertext is the concatenation of the initial vector and these individual blocks: $C = IV \cdot C_1 \cdot C_2 \cdots C_l$. CBC mode has been proven to provide strong security guarantees on the privacy of the plaintext message (assuming the underlying block cipher is secure).

Counter Mode: One drawback of CBC mode is that successive blocks must be encrypted sequentially. For high speed applications it is useful to parallelize these computations. This is achieved by encrypting a counter initialized to IV to obtain a sequence that can now be used as though they were the keys for a one-time pad: namely, $Z_i = E_K(IV + i)$ and $C_i = Z_i \oplus M_i$.