Notes for Lecture 1

This course assumes CS170, or equivalent, as a prerequisite. We will assume that the reader is familiar with the notions of algorithm and running time, as well as with basic notions of algebra (for example arithmetic in finite fields), discrete math and probability.

General information about the class, including prerequisites, grading, and recommended references, are available on the class home page.

Cryptography is the mathematical foundation on which one builds secure systems. It studies ways of securely storing, transmitting, and processing information. Understanding what cryptographic primitives can do, and how they can be composed together, is necessary to build secure systems, but not sufficient. Several additional considerations go into the design of secure systems, and they are covered in various Berkeley graduate courses on security.

In this course we will see a number of rigorous definitions of security, some of them requiring seemingly outlandish safety, even against entirely implausible attacks, and we shall see how if any cryptography at all is possible, then it is also possible to satisfy such extremely strong notions of security. For example, we shall look at a notion of security for encryption in which an adversary should not be able to learn any information about a message given the ciphertext, even if the adversary is allowed to get encodings of any messages of his choice, and *decodings* of any ciphertexts of his choices, with the only exception of the one he is trying to decode.

We shall also see extremely powerful (but also surprisingly simple and elegant) ways to define security for protocols involving several untrusted participants.

Learning to think rigorously about security, and seeing what kind of strength is possible, at least in principle, is one of the main goals of this course. We will also see a number of constructions, some interesting for the general point they make (that certain weak primitives are sufficient to make very strong constructions), some efficient enough to have made their way in commercial products.

1 Alice, Bob, Eve, and the others

Most of this class will be devoted to the following simplified setting: Alice and Bob communicate over an insecure channel, such as the internet or a cell phone. An eavesdropper, Eve, is able to see the whole communication and to inject her own messages in the channel.

Alice and Bob hence want to find a way to encode their communication so as to achieve:

- **Privacy:** Eve should have no information about the content of the messages exchanged between Alice and Bob;
- **Authentication:** Eve should not be able to impersonate Alice, and every time that Bob receives a message from Alice, he should be sure of the identity of the sender. (Same for messages in the other direction.)

For example, if Alice is your laptop and Bob is your wireless router, you might want to make sure that your neighbor Eve cannot see what you are doing

on the internet, and cannot connect using your router.

For this to be possible, Alice and Bob must have some secret information that Eve ignores, otherwise Eve could simply run the same algorithms that Alice does, and thus be able to read the messages received by Alice and to communicate with Bob impersonating Alice.

In the classical *symmetric-key cryptography* setting, Alice and Bob have met before and agreed on a secret *key*, which they use to encode and decode message, to produce authentication information and to verify the validity of the authentication information.

In the *public-key* setting, Alice has a *private key* known only to her, and a *public key* known to everybody, including Eve; Bob too has his own private key and a public key known to everybody. In this setting, private and authenticated communication is possible without Alice and Bob having to meet to agree on a shared secret key.

This gives rise to four possible problems (symmetric-key encryption, symmetric-key authentication, public-key encrpytion, and public-key authentication, or *signatures*), and we shall spend time on each of them. This will account for more than half of the course.

The last part of the course will deal with a fully general set-up in which any number of parties, including any number of (possibly colluding) bad guys, execute a distributed protocol over a communication network.

In between, we shall consider some important protocol design problems, which will play a role in the fully general constructions. These will be *commitment schemes*, *zero-knowledge proofs* and *oblivious transfer.*

2 The Pre-history of Encryption

The task of encoding a message to preserve privacy is called *encryption* (the decoding of the message is called *decrpytion*), and methods for symmetric-key encryption have been studied for literally *thousands* of years.

Various *substitution* ciphers were invented in cultures having an alphabetical writing system. The secret key is a permutation of the set of letters of the alphabet, encryption is done by applying the permutation to each letter of the message, and decryption is done by applying the inverse permutation. Examples are

- the Atbash ciphers used for Hebrew, in which the first letter of the alphabet is replaced with the last, the second letter with the second-to-last, and so on. It is used in the book of Jeremiah
- the cipher used by Julius Caesar, in which each letter is shifted by three positions in the alphabet.

There are reports of similar methods used in Greece. If we identify the alphabet with the integers $\{0, \ldots, k-1\}$, where k is the size of the alphabet, then the Atbash code is the mapping $x \rightarrow k-1-x$ and Caesar's code is $x \rightarrow x+3 \bmod k$. In general, a substitution code of the form $x \rightarrow x+i \bmod k$ is trivially breakable because of the very small number of possible keys that one has to try. Reportedly, former Mafia boss Bernardo Provenzano used Caesar's code to communicate with associates while he was a fugitive. (It didn't work too well for him.)

The obvious flaw of such kind of substitution ciphers is the very small number of possible keys, so that an adversary can simply try all of them.

Substitution codes in which the permutation is allowed to be arbitrary were used through the middle ages and modern times. In a 26-letter alphabet, the number of keys is 26!, which is too large for a brute-force attack. Such systems, however, suffer from easy total breaks because of the facts that, in any given language, different letters appear with different frequencies, so that Eve can immediately make good guesses for what are the encryptions of the most common letters, and work out the whole code with some trial and errors. This was noticed already in the 9th century A.D. by Arab scholar al-Kindy. Sherlock Holmes breaks a substitution cipher in *The Adventure of the Dancing Men.*

For fun, try decoding the following message. (A permutation over the English alphabet has been applied; spaces have been removed before encoding.)

IKNHQHNWKZHTHNHPZKTPKAZYASNKOOAVHNPSAETKOHQHNCH
HZSKBZRHYKBRCBRNHIBOHYRKCHXZKSXHYKBRAZYIKNHQHNWK
ZHETKTAOORBVCFHYCBRORKKYNPDTRCASXBLAZYIKNHQHNWK

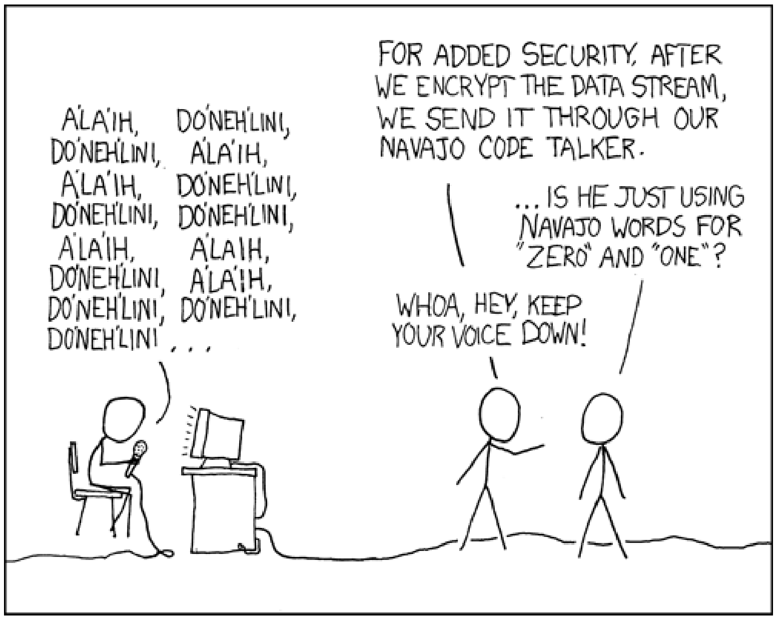
ZHETKEKNXOTANYAZYZHQHNDPQHOBLRTPOKZHPOIKNWKBWKB
XZKEETARRTHWOAWAOKTPKDKHOOKDKHORTHZARPKZEHFFRT
POZARPKZOSKVPZDCASXAZYOKPORTPOSAVLAPDZRTHLHKLFHK
IKTPKTAQHOAPYPRFKBYFWAZYSFHANFWEHNHDKPZDKZEHNHDK
PZDORNKZDAZYEHNHDKPZDAFFRTHEAWWKBXZKERTHWSAFFKT
PKACHFFEHRTHNORARHPROACARRFHDNKBZYORARHPROAORA
RHRTARXZKEOTKERKLPSXALNHOPYHZRAZYZKSAZYPYARHPZNH
SHZRTPORKNWYHVKSNARKNNHLBCFPSAZTAOEKZRTHETPRHTK
BOHEPRTKBREPZZPZDRTHKTPKLNPVANW

Other substitution ciphers were studied, in which the code is based on a permutation over Σ^t, where Σ is the alphabet and t a small integers. (For example, the code would specify a permutation over 5-tuples of characters.) Even such systems suffer from (more sophisticated) frequency analysis.

Various tricks have been conceived to prevent frequency analysis, such as changing the permutation at each step, for example by combining it with a cyclic shift permutation. (The German Enigma machines used during WWII used multiple permutations, and applied different shift on each application.)

More generally, however, most classic methods suffer from the problem of being *deterministic* encryption schemes: If the same message is sent twice, the encryptions will be the same. This can be disastrous when the code is used with a (known) small set of possible messages

This xkcd cartoon makes this point very aptly.


ÁLA'IH, DO'NEH'LINI, DO'NEH'LINI, ÁLA'IH, ÁLA'IH, DO'NEH'LINI, DO'NEH'LINI, DO'NEH'LINI, ÁLA'IH, ÁLA'IH, DO'NEH'LINI, ÁLA'IH, DO'NEH'LINI, DO'NEH'LINI, DO'NEH'LINI . . .
FOR ADDED SECURITY, AFTER WE ENCRYPT THE DATA STREAM, WE SEND IT THROUGH OUR NAVAJO CODE TALKER.
... IS HE JUST USING NAVAJO WORDS FOR "ZERO" AND "ONE"?
WHOA, HEY, KEEP YOUR VOICE DOWN!

(The context of the cartoon is that, reportedly, during WWII, some messages were encrypted by translating them into the Navajo language, the idea being that there was no Navajo speaker outside of North America. As the comic shows, even though this could be a very hard permutation to invert without the right secret information, this is useless if the set of encrypted messages is very small.)

Look also at the pictures of the two encodings of the Linux penguin on the Wikipedia page on block ciphers.

Here is an approach that has large key space, which prevents single-character frequency analysis, and which is probabilistic.

Alice and Bob have agreed on a permutation P of the English alphabet $\Sigma = \{A, \ldots, Z\}$, and they think of it as a group, for example by identifying Σ with $\mathbb{Z}/26\mathbb{Z}$, the integers mod 26.

When Alice has a message $m_1 \cdots m_k$ to send, she first picks a random letter r, and then she produces an encryption $c_0, c_1, \ldots, c_k$ by setting $c_0 = r$ and $c_i := P(c_{i-1} + m_i)$. Then Bob will decode $c_0, \ldots, c_k$ by setting $m_i := P^{(-1)}(c_i) - c_{i-1}$.

Unfortunately, this method suffers from *two-character* frequency analysis. You might try to amuse yourselves by decoding the following ciphertext (encoded with the above described method):

HTBTOOWCHEZPWDVTBYQWHFDBLEDZTESGVFO
SKPOTWILEJQBLSOYZGLMVALTQGVTBYQPLHAKZ
BMGMGDWSTEMHNBVHMZXERHJQBEHNKPOMJDP
DWJUBSPIXYNNRSJQHAKXMOTOBIMZTWEJHHCFD
BMUETCIXOWZTWFIACZLRVLTQPDBDMFPUSPFYW
XFZXXVLTQPABJFHXAFTNUBBJSTFHBKOMGYXGKC
YXVSFRNEDMQVBSHBPLHMDOOYMVWJSEEKPILOB
AMKMXPPTBXZCNNIDPSNRJKMRNKDFQZOMRNFQZ
OMRNF

As we shall see later, this idea has merit if used with an *exponentially big* permutation, and this fact will be useful in the design of actual secure encryption schemes.

3 Perfect Security and One-Time Pad

Note that if Alice only ever sends one one-letter message m, then just sending $P(m)$ is completely secure: regardless of what the message m is, Eve will just see a random letter $P(m)$. That is, the distribution (over the choice of the secret key P) of encodings of a message m is *the same* for all messages m, and thus, from the point of view of Eve, the encryption is *statistically independent* of the message.

This is an ideal notion of security: basically Eve might as well not be listening to the communication, because the communication gives no information about the message. The same security can be obtained using a key of $\log 26$ bits (instead of $\log 26!$ as necessary to store a random permutation) by Alice and Bob sharing a random letter r, and having Alice send $m + r$.

In general, is Alice wants to send a message $m \in \Sigma^k$, and Alice and Bob share a random secret $r \in \Sigma^k$, then it is perfectly secure as above to send $m_1 + r_1, \ldots, m_k + r_k$.

This encoding, however, can be used *only once* (think of what happens when several messages are encoded using this process with the same secret key) and it is called *one-time pad*. It has, reportedly, been used in several military and diplomatic applications.

The inconvenience of one-time pad is that Alice and Bob need to agree in advance on a key as large as the total length of all messages they are ever going to exchange. Obviously, your laptop cannot use one-time pad to communicate with your base station.

Shannon demonstrated that perfect security requires this enormous key length. Without getting into the precise result, the point is that if you have an n-bit message and you use a k-bit key, $k < n$, then Eve, after seeing the ciphertext, knows that the original message is one of 2^k possible messages, whereas without seeing the ciphertext she only knew that it was one of 2^n possible messages.

When the original message is written, say, in English, the consequence of short key length can be more striking. English has, more or less, one bit of entropy per letter which means (*very* roughly speaking) that there are only about 2^n meaningful n-letter English sentences, or only a $(1/13)^n$ fraction of all $(26)^n$ possible n-letter strings. Given a ciphertext encoded with a k-bit key, Eve knows that the original message is one of 2^k possible messages. Chances are, however, that only about $2^k \cdot (13)^{-n}$ such messages are meaningful English sentences. If k is small enough compared to n, Eve can uniquely reconstruct the original message. (This is why, in the two examples given above, you have enough information to actually reconstruct the entire original message.)

When $n >> k$, for example if we use an 128-bit key to encrypt a 4GB movie, virtually all the information of the original message is available in the encryption. A brute-force way to use that information, however, would require to try all possible keys, which would be infeasible even with moderate key lengths. Above, we have seen two examples of encryption in which the key space is fairly large, but efficient algorithms can reconstruct the plaintext. Are there always methods to *efficiently* break any cryptosystem?

We don't know. This is equivalent to the question of whether one-way functions exist, which is probably an extremely hard question to settle. (If, as believed, one-way functions do exist, proving their existence would imply a proof that $P \neq NP$.)

We shall be able, however, to prove the following dichotomy: either one-way functions do not exist, in which case any approach to essentially any cryptographic problem is breakable (with exceptions related to the one-time pad), or one-way functions exist, and then all symmetric-key cryptographic problems have solutions with extravagantly strong security guarantees.

Next, we'll see how to formally define security for symmetric-key encryption, and how to achieve it using various primitives.