Computer Science 276: Cryptography
Lecture 1

Professor David Wagner*

21 January 2004

I hate beginning a course with boring administrivia, so I won't. You can find the syllabus, requirements, grading policy, and other useful information on the course webpage,

`http://www.cs.berkeley.edu/~daw/teaching/cs276-s04/`.

I will, however, say something about what material this course covers. We can look at cryptography in the form of a layer cake (or whatever you'd prefer to call this kind of diagram):

design and implementation of secure systems
modes of operation; protocols
crypto primitives: RSA, DES, etc.

The top layer is the subject of CS 261, while the bottom layer has in the past been the topic of a CS 294 seminar. This course is about the middle layer: how do we take crypto primitives and build useful things out of them?

Before we begin that task, let's go back to the very beginning.

A Brief History of Cryptography

Cryptography has been around for over two thousand years; one date commonly cited for the earliest cryptosystems is 500 BC. The Romans provide

*Scribe notes by David Freeman, `dfreeman@math.berkeley.edu`.

some good examples of early efforts to send secret messages, some of which were not cryptosystems as we conceive of them today. For example, they would shave a slave's head, write the message to be transmitted on the exposed skin, wait for the hair to grow back, and send the slave off to deliver the message. If the slave fell into enemy hands, the enemy wouldn't find any message (unless they knew where to look). If the slave made it through, the recipient would shave the slave's head and read the message. This system isn't too useful for sending frequent messages, as the latency is rather high.

Another example of Roman cryptography is the so-called "Caesar cipher." This consists of shifting each letter in the alphabet forward by a fixed amount (wrapping around mod 26 if necessary). One example popular on Usenet is RA-13, which shifts each letter foward by 13. [RA-13 encoded message shown on overhead.] As you can see, it's pretty easy to decrypt a RA-13-encoded message (in fact, the standard Usenet reader has RA-13 built in), so it's not too secure for sending secret messages.

A final example of early cryptography is the "Cryptoquote," which is a simple substitution cipher. Mathematically, we may think of such a cipher as a permutation on the 26 letters. [Cryptoquote shown on overhead.] Let's think about how to decrypt this message. We can look at patterns in the words; a one-letter word has to be 'a' or 'I'; similarly, the number of two-letter words is limited. Certain letters are repeated, which could give clues. We can use a frequency table: e is the most common letter in English, followed by t, so if our cryptogram has more x's than any other letter it's a safe bet that x stands for e. If we have some more technology at our disposal we can do more advanced pattern searches. In fact, we note that this message has an eleven-letter word where the first three letters are repeated in the sixth through eighth letters. Doing an egrep search on the dictionary produces only three words with this property: *amalgamates, amalgamated,* and *mathematics.* I'll leave you to guess which one it is.

The cryptoquote example shows that the more one knows about the English language, the easier it is to decrypt the message. This helps explain why the earliest cryptographyers were mostly linguists. It was only around 1900 that people began to realise that mathematics and statistics (e.g. the alphabet frequency table) might be able to help out.

The systems described above have all been examples of the first phase of cryptographic history, what we may call the "pencil and paper era." Codes devised during this period were simple and messages could be encoded and decoded by hand. And as we have seen, the codes were not very secure.

Perhaps the most famous message sent during the first era of cryptography is the Zimmerman Telegram. This telegram was sent from the German Foreign Minister, Arthur Zimmerman, to the German legation in Mexico. The telegram declared that Germany was about to begin unrestricted submarine warfare, and proposed an alliance with Mexico. The United States intercepted the telegram and, thanks to a captured codebook, decoded it and made it public. The official and public reaction helped lead to America's entry into the first World War.

Chagrined by the Zimmerman telegram and determined to get it right in time for the next war, the Germans established a project to create a mechanical device that would encrypt a message in an unbreakable code. Thus was ushered in the second phase of cryptography, what we may call the "mechanical era." The Germans' result was the Enigma machine, which was extremely advanced and should have been unbreakable. But (and there's always a but) a few Polish mathematicians, rather irked at the Germans tromping through their country for the second time in 25 years, somehow (perhaps through bribery) obtained a working Enigma machine. This machine eventually found its way to Britain, where the government realised what a devestating effect it could have on the war effort and hired 10,000 people to break the code. They did indeed break the code, and the Germans never suspected that their messages were being read. The resulting leap in intelligence, according to most experts, shortened the European war by about a year. (In fact, the British never let anyone know they had cracked the code until the last working Enigma machine went out of service, sometime around 1970.)

Another code the Germans used around this time was called Tunny. It worked using a pseudorandom generator; some kind of seed produced a keystream ks. This keystream was exclusive-ored with the plain text p to produce a cipher c:

$$c = p \oplus ks.$$

How did the Allies crack this code? Again there was a stroke of luck. At some point the Germans sent a coded message, but it was received garbled, and the recipient sent back a message, in the clear, asking the sender to retransmit. The sender, annoyed and in a hurry, retransmitted the message, but made some abbreviations. The result was two messages that were identical for the first bunch of characters, and then diverged wildly. The Allies made the guess that in the second message a character had been deleted. If the first message was

$$c = p \oplus ks$$

and the second message was

$$c' = (p \ll 1) \oplus ks,$$

then we have

$$c \oplus c' = p \oplus (p \ll 1).$$

Thus if the first character of the plaintext p is known (or guessed), then all of p can be recovered from $c \oplus c'$. This information can be used to recover the keystream ks, and the keystream can be analysed to discover how it is generated, and thus how to decode other Tunny messages.

The Germans were not the only Axis power to have their codes broken by the Allies; the Japanese were also susceptible. The most famous example involves a message that the United States decrypted in 1942 that described a planned attack on a target referred to only as "AF." The only problem was that the Americans didn't know what "AF" referred to. Some believed it was Midway Island; to check this hypothesis Midway's commanding officer was instructed to send a message (in the clear) that the island's desalinisation plant had broken down and needed immediate repairs. Soon thereafter, the Navy intercepted a message that said "AF is short of water." Now that they knew the target, the Americans could prepare for the attack on Midway, which turned out to be a turning point in the Pacific war.

We can take two important lessons from these examples of wartime codebreaking. The first lesson is that paranoia really counts. Neither the Japanese nor the Germans suspected that their codes had been broken, so they continued to use them to send their messages, which was a great boon to Allied intelligence. The second lesson is Kirchoff's law (sp?): all security rests in the key. Both the Germans and the Japanese relied on the secrecy of the encryption algorithm, and because of it their codes were doomed.

The first major theoretical development in cryptography after World War II was Claude Shannon's development of information theory, which he then applied to certain codes. In particular, Shannon gave an information-theoretic analysis of the one-time pad code.

The modern era of cryptography was ushered in during the 1970s. Banks had started using computers in their business, and they wanted digital encryption standards. They went to NIST, which worked for a while and came back with DES – the Data Encryption Standard. This is where our story ends and our course begins.

What is Security?

Before we can try to figure out what it means for an encryption scheme to be secure, we must first define the concept of an encryption scheme.

Definition. *An encryption scheme is a triple of (possibly randomised) algorithms* $(\mathrm{Gen}, \mathrm{E}, \mathrm{D})$ *with*

$$\begin{aligned} \mathrm{Gen}() &\to Keys \\ \mathrm{E} : Keys \times \mathcal{M} &\to \mathcal{C} \\ \mathrm{D} : Keys \times \mathcal{C} &\to \mathcal{M}, \end{aligned}$$

where $Keys$ *is the space of possible keys,* $\mathcal{M}$ *is the space of possible messages, and* $\mathcal{C}$ *is the space of possible ciphertexts.*

Furthermore, this triple must satisfy

$$\mathrm{D}(k, \mathrm{E}(k, m)) = m$$

for all $k \in Keys$ *and* $m \in \mathcal{M}$.

In this definition, Gen is the key generation algorithm (usually a random variable distributed uniformly on the keyspace $Keys$), E is the encryption algorithm, and D is the decryption algorithm. The requirement $\mathrm{D}(k, \mathrm{E}(k, m)) = m$ ensures that we can decrypt any message if we know the key.

To give us a picture of this situation, we introduce two notorious characters in the story of cryptography, Alice and Bob. Alice has a message m she wants to send to Bob securely. She sends Bob the ciphertext $c = \mathrm{E}(k, m)$, and Bob deciphers the message using $m = \mathrm{D}(k, c)$. If the system is secure, then an eavesdropper not knowing the key k cannot gain any information just from reading the ciphertext c. But what exactly does this mean?

As our first example we will use the "one-time pad" code. In this code, the spaces of keys, messages, and ciphertexts are l-bit strings:

$$Keys = \mathcal{M} = \mathcal{C} = \{0,1\}^l.$$

We encode a message by exclusive-oring the key with the message:

$$\mathrm{E}(k, m) = k \oplus m.$$

We do the same thing to decrypt a ciphertext:

$$\mathrm{D}(k, c) = c \oplus m.$$

In this system, any given key can only be used for one message, or else security is compromised; hence the "one-time pad."

How secure is this system? Well, there are many different ways we can define security. Here are four possibilities:

1. A message is secure if the probability that an eavesdropper guesses the message is zero. Since our messages are all of finite length, this is impossible (there is always at least a very small chance of guessing the message), so this is a bad criterion.

2. A message is secure if the probability that an eavesdropper can guess the message given the ciphertext is no better than the probability of guessing the message at random. In our example, this is

   $$\mathrm{Prob}_{K,M}(M = m | \, \mathrm{E}(K, M) = c) = 2^{-l}$$

   for all $m \in \mathcal{M}$, $c \in \mathcal{C}$. (M is a random variable representing the message sent, m is the message guessed, and c is the ciphertext.) This is also not a good system; what if the only two possibilities for the message are "Yes" and "No," but the ciphertext is many bits long? If the probability of guessing the message is no better than $1/2$ then the message should be considered secure, which it is not under this criterion.

3. A message is secure if the probability of guessing the message knowing the ciphertext is the same as the probability of guessing the message without knowing the ciphertext. In our example, this is

   $$\mathrm{Prob}_{K,M}(M = m | \, \mathrm{E}(K, M) = c) = \mathrm{Prob}_{K,M}(M = m)$$

for all $m \in \mathcal{M}$, $c \in \mathcal{C}$. This is called *Shannon secrecy* with respect to the random variable M.

4. A message is secure if the probability that any two messages give the ciphertext c is the same. Namely,

$$\text{Prob}_K(E(K, m_0) = c) = \text{Prob}_K(E(K, m_1) = c)$$

for all $m_0, m_1 \in \mathcal{M}$. This is called *perfect secrecy.*

Perfect secrecy and Shannon secrecy are two good ways to measure security. In fact, we will prove the following in the next lecture:

Theorem 1. *Perfect secrecy is equivalent to Shannon secrecy with respect to all random variables.*

Given this theorem, it is easy to prove that the one-time pad is secure.

Theorem 2. *The one-time pad has perfect secrecy.*

Proof. If the k is uniformly randomly distributed in the keyspace $\{0,1\}^l$, then

$$\text{Prob}(m_0 \oplus k = c) = \text{Prob}(m_1 \oplus k\ c) = 2^{-l}$$

since all k are equally likely. □

As a final remark, we note that these definitions of security treat each message individually, and ignore any information an eavesdropper may glean from picking up several consecutive messages.