Professor M. J. Fischer

January 11 & 13, 2005

Lecture Notes, Week 1

1 Course Overview

This course is about cryptography and its application to computer and network security. A better term might be *information security*, which includes all forms of information protection, whether the information resides in a computer, on the network, or on a storage device such as a hard disk, DVD, or even paper. Security is a huge field and includes important non-technical topics such as incentive, opportunity, deterrence, threat, trust, anonymity, and coercion, all of which play together into the overall security picture. Our focus is on a particular set of useful tools—cryptographic primitives—that that can be used to enhance information security. Nevertheless, we need some awareness of general security issues in order to understand the environment in which cryptographic tools are used and to motivate the properties that we desire of these tools.

2 Organization

The course will be roughly organized around *cryptographic primitives*, objects having properties that make them useful in solving certain security problems. For each such primitive, one can ask:

What can be done with it? This leads to a study of cryptographic algorithms and protocols.

What are its properties? This leads to modeling and analysis, which very quickly requires complexity theory, probability theory, and statistics.

How is it built? This requires a fair amount of mathematics, particularly number theory and algebra. Don't worry if you haven't studied these topics before. We will cover what is need for the methods discussed in this course.

How is it implemented? Implementing cryptographic primitives requires a lot of attention to detail, especially to make sure that the programs don't accidentally leak secret information. This course will involve some implementation.

3 Example primitive: Symmetric cryptography

A *symmetric cryptosystem* (sometimes called a *private-key* or *one-key* system) is a pair of functions E and D such that $D(k, E(k, m)) = m$ for all keys k and all messages m. Moreover, given $c = E(k, m)$, it is hard to find m without knowing the key k.

3.1 What can be done with a symmetric cryptosystem?

This is the classical tool for solving the *secret message transmission problem*:

1. Alice wants to send Bob a private message m over the internet.

2. Alice and Bob both have a secret key k.

3. Alice computes $c = E(k, m)$ and sends c to Bob.

4. Bob receives c', computes $m' = D(k, c')$, and assumes m' to be Alice's message.

We assume that the network is insecure and that an eavesdropper, Eve, can listen in and learn c. We desire nonetheless that m remain private.

3.2 What do we require of E, D, and the computing environment in order for this protocol to accomplish Alice and Bob's goals?

- Given c, it is hard to find m without also knowing k.
- k is not initially known to Eve.
- Eve can guess k with at most an extremely tiny probability of success. This means k must be chosen randomly.
- Alice and Bob protect k from being released. Their computers have not been compromised. Eve is not able to obtain k by looking on Alice's or Bob's hard disk, swap file, or free memory pages, even if she is a legitimate user of that computer or the services it provides.
- There are no unanticipated ways by which Eve succeeds in obtaining k, e.g., social engineering, using binoculars to watch Alice or Bob's keyboard, etc. etc.

4 Information Security

Last time we looked at the secret message transmission problem, which concerned keeping a private message secret from Eve. Information security is certainly partly about keeping data private, but it is also about preventing hackers from breaking into a computer, preventing denial of service attacks against web servers, preventing unauthorized modification of databases, preventing illegal copying of data, preventing fraud in e-commerce applications, and so forth.

Security certainly seems to mean preventing bad things from happening. However, it is not always so easy to decide exactly what are the bad things that one is trying to prevent. Without knowing exactly *what* is to be prevented, one can never how how *effective* a security system is at preventing them. Of course, one also wants to allow those activities to proceed normally that are not proscribed by the security policy.

4.1 A security example from real life

A familiar example of a security problem is that of keeping intruders out of my house.

What do I want to keep out? Specifying exactly what is to be kept out and what allowed is already not such an easy problem. At first sight, I might say that I don't want anybody entering whom I have not specifically authorized. But even this can be a bit problematical.

1. Do I also want to prevent chimpanzees from entering?

2. Do I want to prevent mice from entering?

3. Do I want to prevent the twin brother of my friend Alex from entering (who looks so much like Alex that even I can't always tell them apart)?

While these questions may sound silly, they have serious analogs in information security. To (1), you'd likely say, "Of course, I don't want to let chimpanzees in the house either, but I don't consider them a threat because there are no chimpanzees in my neighborhood." Yet the world of information security is replete with examples of security holes that are of no consequence until someone develops an exploit, at which time they suddenly become serious problems. To (2), you might say, "No, I don't want mice in, either", but at the same time you'd realize that the means of preventing mice from entering are quite different from those of protecting against human intruders, and conflating the two problems will only make finding workable solutions that much harder. To (3), you're forced to think about the meaning of identity. What is it about Alex that makes me want to let him in but not his twin brother? What hope do I have of solving this problem if I can't distinguish between the two? In fact, the simple solution of giving Alex a key to the house works in practice, as long as I know that it is Alex getting the key and not his brother. Distinguishing among individuals becomes a serious problem on the internet, and schemes purporting to identify individuals generally identify instead possession of secret information (a cryptographic key) or a particular object (such as a smart card) instead, with the assumption that this is sufficient to identify the individual. It isn't always so in the real world, for secrets can be stolen and smart cards forged. The rapid growth of identity theft today is evidence of this fact. (See http://www.consumer.gov/idtheft/ for further information on this topic.)

What are some possible means to keep out intruders, and how effective are they? We now look at some of the means that might be used to keep intruders out and discuss their effectiveness.

1. Post a "no trespassing – do not enter" sign.
2. Lock the front door.
3. Install a burglar alarm.
4. Buy a gun.
5. Call the police.
6. Sue the intruder.
7. Conceal the entrance.

Think carefully about each of these means and ask yourself under what conditions it might be effective, and what else must be in place for it to work. For example, suing the intruder presupposes that he can be identified, that you have evidence that he entered illegally, and that society has an effective judicial system. Buying a gun will only deter breakins if it is known that you are likely to possess a gun. Locking the front door has little effect unless the back door and windows are also locked.

4.2 Cryptography and security

Cryptography is to information security as locks are to personal security.

- Both are clever mechanisms that can be analyzed in isolation.

- Both can be effective when used in suitable contexts.
- Both comprise only a small part of the security picture.

4.3 Information security in the real world

Here are just some of the many goals of information security.

- Protection against data damage.
- Protection against theft of intellectual property.
- Protection against surveillance.
- Protection against unauthorized actions.
- Protection of constitutional privacy rights.
- Protection of freedom of information.

How is security achieved in the real world?

- **Prevention:** Physical barriers, locks, encryption, firewalls, etc.
- **Detection:** Audits, checks and balances.
- **Legal means:** Laws, sanctions.
- **Concealment:** Camouflage, steganography.

Absolute security in the real world is infeasible. The goal is risk management. The same is true with information security. Security mechanisms have a cost. One needs to weight their cost against the benefits.

5 Classical cryptography

A *symmetric cryptosystem* consists of a set $\mathcal{M}$ of *plaintext messages*, a set $\mathcal{C}$ of *ciphertexts*, a set $\mathcal{K}$ of keys, and two functions, an *encryption* function $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ and a *decryption* function $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$. We often write $E_k(m) = E(k, m)$ and $D_k(c) = D(k, c)$ and may talk about a cryptosystem consisting of a family of encryption and decryption functions indexed by k, but this should not obscure the fact that in order to be useful, it must be feasible to evaluate the functions given an arbitrary key k.

We require that D_k be the (one-sided) inverse of E_k, that is, for all keys $k \in \mathcal{K}$ and all plaintext messages $m \in \mathcal{M}$, we have $D_k(E_k(m)) = m$.

To be useful, it should certainly be difficult to find k given only $c = E_k(m)$. But we generally want much more. The goal is to hide m, not k, so a procedure that found m given c would compromise the cryptosystem even if k remained well hidden. Thus, we want the system to resist a *ciphertext only* attack: the attacker knows c and tries to find m.

In practice, this also might not be enough. The attacker might know a number of pairs plaintext-ciphertext pairs $(m_1, c_1), \ldots, (m_r, c_r)$ where $c_i = E_k(m_i)$ for $i = 1, \ldots, r$, in addition to knowing c. We want it to be difficult for the attacker to find m in this case also. Thus, we want the system to resist a *known plaintext* attack.

In many contexts, one needs even stronger forms of attack resistance, where the attacker not only knows a number of plaintext-ciphertext pairs but also gets to choose the plaintext or the cipher text, or both. Where the attacker gets to experiment with the encryption system, it should still be difficult to find the plaintext corresponding to any ciphertext c that differs from those in the known plaintext-ciphertext pairs. These so-called *chosen plaintext*, *chosen ciphertext*, and *chosen text* attacks are much stronger than the simple ciphertext only attack and much harder to defend against. For example, in a chosen ciphertext attack, the attacker would be allowed to obtain the decryption of $c + 1$ when trying to decrypt c.

5.1 Caeser cipher

The Caeser cipher is said to go back to Roman times. Let the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ consist of strings of uppercase letters from the Roman alphabet and spaces. Number the letters of the alphabet starting at 0, so $A = 0, B = 1, \ldots, Z = 25$. The key space $\mathcal{K} = \{0, \ldots, 25\}$. To encrypt, E_k replaces each letter m by the letter $(m + k) \bmod 26$.[1] To decrypt, D_k replaces each letter c by the letter $(c - k) \bmod 26$.

Consider the problem of breaking the Caeser cipher. Suppose you intercept the ciphertext JXQ. You quickly discover that $E_3(\text{GUN}) = \text{JXQ}$. But can you conclude that $k = 3$ and GUN is the corresponding plaintext? Upon further investigation, you discover that $E_{23}(\text{MAT}) = \text{JXQ}$. So now you are in a quandary. You have two plausible decryptions and no way to tell for sure which is correct. Have you broken the system or haven't you? According to the "definition" above, you haven't since you haven't found the plaintext with certainty. But you've certainly inflicted serious damage. You've reduced the possible set of 3-letter words that the message might have been down to a small set of possibilities. In many contexts, this is nearly as good as having completely broken the system. This suggests that a formal definition of security must take into account the possibility of "partially breaking" a system, as we did here.

The longer the message, the more likely that only one key will lead to a sensible message. For example, if the ciphertext were "EXB JXQ", you would know that $k = 3$, since $D_3(\text{EXB JXQ}) =$ BUY GUN, whereas $D_{23}(\text{EXB JXQ}) =$ HAE MAT is nonsense.

At the opposite extreme, consider the Caeser cipher with a one-letter message. If I tell you that the ciphertext is V, you learn absolutely nothing about the plaintext. You reason as follows: If $k = 0$, then $m =$ V. If $k = 1$, then $m =$ U. If $k = 2$, then $m =$ T. And so on through the alphabet. For every possible plaintext message m, there is exactly one key for which m encrypts to V.[2] So knowing that $c =$ V doesn't give the attacker any information about the plaintext m.

We say such a system is *information-theoretically secure*. This means that there is no statistical correlation between the plaintext and the ciphertext. To make this definition precise, one must assume that the key k is chosen uniformly at random and is unknown to the attacker. What the attacker sees is a random variable $c = E_k(m)$, where m is a constant and k a uniformly distributed random variable. c is identically distributed for each choice of m; hence, the distribution of c does not depend on m, so knowing c gives no information about m.

Thus, we see that the same Caeser cipher can be information-theoretically secure, or partially secure, or completely breakable, depending on the length of the message to which it is applied.

[1] $x \bmod 26$ is the remainder of x divided by 26.

[2] Why does it matter that there is exactly one?

6 Unbreakable Codes

A code can be hard for Eve to break for two different reasons: (i) She may not have enough information to uniquely determine the plaintext, or (ii) she may have enough information, but it takes her too much time to find the plaintext. In the first case, we say the code is *information-theoretically secure*. In the second, it is *computationally secure*. The first notion is based on statistical inference; the second on computational complexity.

6.1 Statistical inference

Statistics is about guessing underlying parameter values from samples drawn from a random distribution. For example, given a biased coin with probability p of heads, one wants to determine p by observing the outcomes of several coin flips. Of course, one can never determine p exactly, so one tries instead to guess a value p' that is "likely" to be "close" to p.

Eve's job in attempting to break a cryptosystem is analogous to that of the statistician. The analog of a coin flip is for Alice to choose a random key k and compute $c = E_k(m)$. Eve tries to guess a value m' that is "likely" to be "close" to m. Unlike the coin-flip example, Eve generally will only get one sample for a given message m. Neverless, the same notions apply, and the closer Eve can get to finding the true message m', and the more likely she is to be correct, the weaker our cryptosystem is.

The strongest case of security is where Eve can get no information at all about m. This case obtains when c is completely independent of m, that is, when the distribution of c does not depend on m. Recall that a (discrete) probability distribution is a function that assigns a probability $p(x)$ to each point x in the sample space. In our case, each value of m results in a potentially different probability distribution $p_m(c)$, where $p_m(c)$ is the probability of choosing a key for which $E_k(m) = c$. To say that the distribution of c does not depend on m is the same as saying that all of the functions $p_m()$ are the same.[3]

For the Caeser cipher with one-letter messages, we have $p_m(c) = 1/26$ for each possible ciphertext letter c. This is because there are 26 equiprobable keys, and each key maps m to a different letter. In this case, c is independent of m, and therefore knowledge of c gives no information about m. We say the system is *information-theoretically secure*.

For the Caeser cipher with two-letter messages, we lose this independence. For example, let $c = \mathtt{VV}$. Then $p_{\mathtt{AA}}(\mathtt{VV}) = 1/26$, but $p_{\mathtt{BC}}(\mathtt{VV}) = 0$. As functions, $p_{\mathtt{AA}}() \neq p_{\mathtt{BC}}()$, and Eve can get information about m from observing c. In particular, Eve knows that m is one of the messages in the set $\{\mathtt{AA}, \mathtt{BB}, \ldots \mathtt{ZZ}\}$.

6.2 One-time pad

A *one-time pad* is a simple information-theoretically secure cryptosystem based on the bitwise *exclusive-or* operator (XOR), which we write as $\oplus$. Recall for Boolean values x and y that $x \oplus y$ is true when exactly one of x and y is true, but is false if x and y are either both false or both true. Exclusive-or is therefore equivalent to sum modulo two, when true is represented by 1 and false by 0. In symbols,

$$x \oplus y = (x + y) \bmod 2.$$

[3]One often assumes that the parameters themselves are also drawn from some distribution. This enables one to talk, for example, about the probability that the ciphertext = "QBA". But it seems somewhat counterintuitive to assume that Alice merely sends random message according to some fixed (but unknown) probability distribution. Therefore, we prefer to avoid such assumptions whenever possible.

With a one-time pad, the plaintext, ciphertext, and key are all assumed to be binary strings of the same length. The encryption function is $E_k(m) = k \oplus m$, where $\oplus$ is applied componentwise to corresponding bits of k and m. It's a basic fact of mod 2 addition that addition and subtraction are the same. This implies that E_k is its own inverse; hence $D_k(c) = E_k(c)$ is the decryption function.

$$D_k(E_k(m)) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0 \oplus m = m.$$

Like the one-letter Caeser cipher, the simple XOR cryptosystem has the property that for every ciphertext string c and every plaintext m, there is exactly one key k such that $E_k(m) = c$ (namely, $m \oplus c$). Hence, every ciphertext is equally likely no matter what m is, so the one-time pad is information-theoretically secure.

The simple XOR cryptosystem would seem to be the perfect cryptosystem. It works for messages of any length (by choosing a key of the same length). It is simple, easy to encrypt and decrypt, and information-theoretically secure. In fact, it is sometimes used for highly sensitive data. However, it suffers from two major weaknesses that make it not so useful in practice:

1. The key k must be as long as the message to be encrypted. Key management and key distribution are both difficult problems which we will be discussing later in the course. The longer the keys, the more difficult these problems become.

2. The same key must never be used more than once. (Hence the term "one-time".) The reason is that the simple XOR cryptosystem immediately succumbs to a known plaintext attack. If Eve knows just one plaintext-ciphertext pair (m_1, c_1), then she can easily compute $k = m_1 \oplus c_1$, totally breaking the system and allowing her to decrypt all future messages sent with that key. Even in a ciphertext-only situation, if Eve has two ciphertexts c_1 and c_2 encrypted by the same key k, she can gain significant partial information about the corresponding plaintexts m_1 and m_2. In particular, she can compute $m_1 \oplus m_2$, since

$$m_1 \oplus m_2 = (c_1 \oplus k) \oplus (c_2 \oplus k) = c_1 \oplus c_2.$$

   That information, together with other information she might have about the likely content of the messages, may be enough for her to seriously compromise the secrecy of the data.

7 Theoretically Breakable Codes

For any fixed message m, the number of possible ciphertexts is at most the size of the key space. (It will be even less if two or more keys map m to the same ciphertext, which is perfectly possible.) Often the message space and the ciphertext space are the same, in which case $E_k(m)$ is a bijection for each fixed value of k. From these two facts, it follows that for each fixed ciphertext c, the number of possible plaintext messages is at most $|\mathcal{K}|$. Hence, knowing the ciphertext c immediately reduces the uncertainty about m from being a member of a set of size $|\mathcal{M}|$ to a member of a set of size $|\mathcal{K}|$. The key spaces of most codes used in practice are much smaller than the space of possible messages and ciphertext. Hence, information-theoretically, Eve gains a lot of information about m just knowing c. Moreover, if the set of meaningful messages is sparse in $\mathcal{M}$ and not correlated with the structure of E_k (the usual case), then it highly likely that the only meaningful message that maps to c is Alice's original message m.

7.1 Brute force attacks

A *brute force attack* is one that tries all possible keys k. For each k, Eve computes $m_k = D_k(c)$ and tests if m_k is meaningful. If so, then m_k is a plausible decryption of c. If exactly one m_k is found that is meaningful, then Eve knows that $m_k = m$.

Given long enough messages, the Caeser cipher is easily broken by brute force—one simply tries all 26 possible keys and sees which leads to a sensible plaintext. The Caeser cipher succumbs because the key space is so small.

With modern computers, it is quite feasible for an attacker to try millions ($\sim 2^{20}$) or billions ($\sim 2^{30}$) of keys. Of course, the attacker also needs some automated test to determine when she has a likely candidate for the real key, but such tests are often easy to produce given a little knowledge of the probable message space.

How big is big enough? The DES (Data Encryption Standard) cryptosystem (which we will talk about shortly) has only 56-bit keys for a key space of size 2^{56}. A special DES Key Search Machine was built as a collaborative project by Cryptography Research, Advanced Wireless Technologies, and EFF. (See http://www.cryptography.com/resources/whitepapers/DES.html for details.) This machine was capable of searching 90 billion keys/second and discovered the RSA DES Challenge key on July 15, 1998, after searching for 56 hours. The entire project cost was under $250,000.

Today, 80-bit keys are probably still safe, but only barely so. 2^{80} is only about 16 million times as large as the DES key space, and tens of thousand of machine on the internet could conceivably be deployed in breaking a code. If not quite feasible using today's PC's (and I'm not saying it isn't), it's not that far-fetched to imagine searching an 80-bit key space in the foreseeable future. Much better are systems like triple DES (with 112-bit keys) and AES (with 128-bit keys), which will probably always be safe from brute-force attacks (but not necessarily from other kinds of attacks).