Security Engineering

Lesson 3
Introduction to Cryptography

Spring 2010
Dr. Marenglen Biba

0011

Foundations of Cryptography

42

Outline

1.1 Terminology

1.2 Steganography

1.3 Substitution Ciphers and Transposition Ciphers

1.4 Simple XOR

1.5 One-Time Pads

1.6 Computer Algorithms

1.7 Large Numbers

Cryptography

- The art and science of keeping messages secure is **cryptography**, and it is practiced by **cryptographers**.

- **Cryptanalysts** are practitioners of **cryptanalysis**, the art and science of breaking ciphertext;

- The branch of mathematics encompassing both cryptography and cryptanalysis is **cryptology** and its practitioners are **cryptologists**.

Terminology

Sender and Receiver

- Suppose a sender wants to send a message to a receiver. Moreover, this sender wants to send the message securely: She wants to make sure an eavesdropper cannot read the message.

Messages and Encryption

- A message is **plaintext** (sometimes called cleartext). The process of disguising a message in such a way as to hide its substance is **encryption**.
- An encrypted message is **ciphertext**. The process of turning ciphertext back into plaintext is **decryption**.
- If you want to follow the ISO 7498-2 standard, use the terms "**encipher**" and "**decipher**." It seems that some cultures find the terms "**encrypt**" and "**decrypt**" offensive, as they refer to dead bodies.

Terminology

- **Plaintext** is denoted by *M*, for message, or *P*, for plaintext.
- It can be a stream of bits, a text file, a bitmap, a stream of digitized voice, a digital video image...whatever.
- As far as a computer is concerned, *M* is simply **binary data**.
- The plaintext can be intended for either transmission or storage. In any case, *M* is the message to be encrypted.
- **Ciphertext** is denoted by *C*. It is also binary data: sometimes the same size as *M*, sometimes larger.
- By combining encryption with compression, *C* may be smaller than *M*. However, encryption does not accomplish this.

Terminology

- The **encryption function *E***, operates on *M* to produce *C*. Or, in mathematical notation:

$$E(M) = C$$

- In the reverse process, the **decryption function *D*** operates on *C* to produce *M*:

$$D(C) = M$$

- Since the whole point of encrypting and then decrypting a message is to recover the original plaintext, the following identity must hold true:

$$D(E(M)) = M$$

Terminology

Authentication, Integrity, and Nonrepudiation

- In addition to providing confidentiality, cryptography is often asked to do other jobs:
  - **Authentication**. It should be possible for the receiver of a message to ascertain its origin; an intruder should not be able to masquerade as someone else.
  - **Integrity**. It should be possible for the receiver of a message to verify that it has not been modified in transit; an intruder should not be able to substitute a false message for a legitimate one.
  - **Nonrepudiation**. A sender should not be able to falsely deny later that he sent a message.

Terminology

Algorithms and Keys

- A **cryptographic algorithm**, also called a **cipher**, is the mathematical function used for encryption and decryption.
  - Generally, there are two related functions: one for encryption and the other for decryption.
- If the security of an algorithm is based on keeping the way that algorithm works a secret, it is a **restricted** algorithm.
- Restricted algorithms have historical interest, but are inadequate by today's standards.
  - A large or changing group of users cannot use them, because every time a **user leaves the group** everyone else must switch to a different algorithm.
  - If someone accidentally **reveals the secret**, everyone must change their algorithm.

Terminology

- Despite the major drawbacks, restricted algorithms are enormously popular for **low-security applications**.
  - Users either don't realize or don't care about the security problems inherent in their system.
- Modern cryptography solves this problem with a **key**, denoted by *K*.
  - This key might be any one of a large number of values. The range of possible values of the key is called the **keyspace**.
  - Both the encryption and decryption operations use this key (i.e., they are dependent on the key and this fact is denoted by the k subscript), so the functions now become:

$$E_K(M) = C$$
$$D_K(C) = M$$

- Those functions have the property that:

$$D_K(E_K(M)) = M$$

Terminology

- Some algorithms use a different encryption key and decryption key. That is, the encryption key, K_1, is different from the corresponding decryption key, K_2. In this case:

$$E_{K1}(M) = C$$

$$D_{K2}(C) = M$$

$$D_{K2}(E_{K1}(M)) = M$$

Terminology

- A **cryptosystem** is an algorithm, plus all possible plaintexts, ciphertexts, and keys.

Symmetric Algorithms

- There are two general types of key-based algorithms: symmetric and public-key.
- **Symmetric algorithms**, **sometimes called conventional algorithms, are algorithms where the encryption key can be calculated from the decryption key and vice versa**.
- In most symmetric algorithms, the encryption key and the decryption key are the same. These algorithms, also called **secret-key** algorithms, **single-key** algorithms, or **one-key** algorithms, require that
  - **The sender and receiver agree on a key before they can communicate securely.**
- The security of a symmetric algorithm rests in the key; divulging the key means that anyone could encrypt and decrypt messages. As long as the communication needs to remain secret, **the key must remain secret**.
- Encryption and decryption with a symmetric algorithm are denoted by:
  - $E_K(M) = C$
  - $D_K(C) = M$

Symmetric Algorithms

- Symmetric algorithms can be divided into two categories. Some operate on the plaintext a single bit (or sometimes byte) at a time; these are called **stream algorithms** or **stream ciphers**.
- Others operate on the plaintext in groups of bits. The groups of bits are called **blocks**, and the algorithms are called **block algorithms** or **block ciphers**.
- For modern computer algorithms, a typical block size is 64 bits—large enough to preclude analysis and small enough to be workable.
- Before computers, algorithms generally operated on plaintext one character at a time. You can think of this as a stream algorithm operating on a stream of characters.

Public-Key Algorithms

- **Public-key algorithms** (also called **asymmetric algorithms**) are designed so that the key used for encryption is different from the key used for decryption.
- Furthermore, the decryption key cannot (at least in any reasonable amount of time) be calculated from the encryption key.
- **The algorithms are called "public-key" because the encryption key can be made public**: A complete stranger can use the encryption key to encrypt a message, but only a specific person with the corresponding decryption key can decrypt the message.
- In these systems, the encryption key is often called the **public key**, and the decryption key is often called the **private key**. The private key is sometimes also called the secret key, but to avoid confusion with symmetric algorithms, that tag won't be used here.

Public-Key Algorithms

- Encryption using public key K is denoted by:

  $E_K(M) = C$

- Even though the public key and private key are different, decryption with the corresponding private key is denoted by:

  $D_K(C) = M$

- Sometimes, messages will be encrypted with the private key and decrypted with the public key; this is used in **digital signatures**.
- Despite the possible confusion, these operations are denoted by, respectively:

  $E_K(M) = C$

  $D_K(C) = M$

Cryptanalysis

- The whole point of cryptography is to keep the plaintext (or the key, or both) secret from eavesdroppers (also called adversaries, attackers, interceptors, interlopers, intruders, opponents, or **simply the enemy**).

- Eavesdroppers are assumed to have complete access to the communications between the sender and receiver.

- **Cryptanalysis is the science of recovering the plaintext of a message without access to the key**.

- Successful cryptanalysis may **recover the plaintext or the key**.
  - It also may find weaknesses in a cryptosystem that eventually lead to the previous results. (The loss of a key through noncryptanalytic means is called a **compromise**.)

Cryptanalysis

- An attempted cryptanalysis is called an **attack**. A fundamental assumption in cryptanalysis, first enunciated by the **Dutchman A. Kerckhoffs** in the nineteenth century, is that the **secrecy must reside entirely in the key**.
- Kerckhoffs assumes that the cryptanalyst has **complete details** of the cryptographic algorithm and implementation. (Of course, one would assume that the CIA does not make a habit of telling Mossad about its cryptographic algorithms, but Mossad probably finds out anyway.)
- While real-world cryptanalysts don't always have such detailed information, it's a good assumption to make.
  - If others can't break an algorithm, even with knowledge of how it works, then they certainly won't be able to break it without that knowledge. ☺

Ciphertext-only attack

- There are four general types of cryptanalytic attacks. Of course, each of them assumes that the cryptanalyst has complete knowledge of the encryption algorithm used:
- **Ciphertext-only attack**. The cryptanalyst has the ciphertext of several messages, all of which have been encrypted using the same encryption algorithm. The cryptanalyst's job is to **recover the plaintext** of as many messages as possible, or better yet to **deduce the key** (or keys) used to encrypt the messages, in order to decrypt other messages encrypted with the same keys.
    - Given: $C_1 = E_k(P_1)$, $C_2 = E_k(P_2)$,...$C_i = E_k(P_i)$ *Deduce: Either* $P_1, P_2,...P_i$; $_k$; *or an algorithm to infer* P_{i+1} *from* $C_{i+1} = E_k(P_{i+1})$

Known-plaintext attack

- The cryptanalyst has access not only to the ciphertext of several messages, but also to the plaintext of those messages. His job is to **deduce the key** (or keys) used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key (or keys).
  - Given: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$,...P_i, $C_i = E_k(P_i)$
  - Deduce: Either k, or an algorithm to infer P_{i+1} from $C_{i+1} = E_k(P_{i+1})$

Chosen-plaintext attack

- The cryptanalyst not only has access to the ciphertext and associated plaintext for several messages, but he also chooses the plaintext that gets encrypted.
- This is more powerful than a known-plaintext attack, because the cryptanalyst can choose specific plaintext blocks to encrypt, ones that might yield more information about the key. His job is to **deduce the key** (or keys) used to encrypt the messages or an **algorithm to decrypt** any new messages encrypted with the same key (or keys).
  - Given: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$,...P_i, $C_i = E_k(P_i)$, where the cryptanalyst gets to choose P_1, P_2,...P_i
  - Deduce: Either k, or an algorithm to infer P_{i+1} from $C_{i+1} = E_k(P_{i+1})$

Adaptive-chosen-plaintext attack

- This is a special case of a chosen-plaintext attack.
- Not only can the cryptanalyst choose the plaintext that is encrypted, but he can also **modify his choice** based on the results of previous encryption.
- In a chosen-plaintext attack, a cryptanalyst might just be able to choose one large block of plaintext to be encrypted; in an adaptive-chosen-plaintext attack he can choose a **smaller block** of plaintext and then **choose another** based on the results of the first, and so forth.

Chosen-ciphertext attack.

- The cryptanalyst can choose different ciphertexts to be decrypted and has **access to the decrypted plaintext**. For example, the cryptanalyst has access to a tamperproof box that does automatic decryption. His job is to **deduce the key**.
  - Given: C_1, $P_1 = D_k(C_1)$, C_2, $P_2 = D_k(C_2)$,...C_i, $P_i = D_k(C_i)$
  - Deduce: k
- This attack is primarily applicable to public-key algorithms.
- A chosen-ciphertext attack is sometimes effective against a symmetric algorithm as well. (Sometimes a chosen-plaintext attack and a chosen-ciphertext attack are together known as a **chosen-text attack**.)

Chosen-key attack

- This attack doesn't mean that the cryptanalyst can choose the key; it means that he has some knowledge about the relationship between different keys.

Rubber-hose cryptanalysis.

- The cryptanalyst threatens, blackmails, or tortures someone until they give him the key. Bribery is sometimes referred to as a **purchase-key attack**.
- These are all very powerful attacks and **often the best** way to break an algorithm.

Kerckhoffs's assumption:

- Don't forget Kerckhoffs's assumption: **If the strength of your new cryptosystem relies on the fact that the attacker does not know the algorithm's inner workings, you're sunk**.
- If you believe that keeping the algorithm's insides secret improves the security of your cryptosystem more than letting the academic community analyze it, you're wrong.
- And if you think that someone won't disassemble your code and reverse-engineer your algorithm, you're naïve. (In 1994 this happened with the RC4 algorithm).
- The best algorithms we have are the ones that have been made public, have been attacked by the world's best cryptographers for years, and are still unbreakable.
  - (The National Security Agency keeps their algorithms secret from outsiders, but they have **the best cryptographers in the world** working within their walls—you don't. Additionally, they discuss their algorithms with one another, relying on **peer review** to uncover any weaknesses in their work.)

Algorithms reliability

- Cryptanalysts don't always have access to the algorithms, as when the United States broke the Japanese diplomatic code PURPLE during World War II—but they often do.
- If the algorithm is being used in a commercial security program, it is simply a **matter of time** and money to disassemble the program and recover the algorithm.
- If the algorithm is being used in a military communications system, it is simply a **matter of time and money** to buy (or steal) the equipment and reverse-engineer the algorithm.
- Those who claim to have an unbreakable cipher simply because they can't break it are either geniuses or fools.
- Beware of people who extol the virtues of their algorithms, but refuse to make them public.
- **Good cryptographers rely on peer review to separate the good algorithms from the bad.**

Security of Algorithms

- Different algorithms offer different degrees of security; it depends on how hard they are to break.

- If the **cost required to break** an algorithm is greater than the value of the encrypted data, then you're probably safe.

- If the **time required to break** an algorithm is longer than the time the encrypted data must remain secret, then you're probably safe.

- If the **amount of data encrypted** with a single key is less than the amount of data necessary to break the algorithm, then you're probably safe.

Categories of breaking

- We have these different categories of breaking an algorithm. In decreasing order of severity:
  - **1. Total break**. A cryptanalyst finds the key, K, such that $D_K(C) = P$.
  - **2. Global deduction**. A cryptanalyst finds an alternate algorithm, A, equivalent to $D_K(C)$, without knowing K.
  - **3. Instance (or local) deduction**. A cryptanalyst finds the plaintext of an intercepted ciphertext.
  - **4. Information deduction**. A cryptanalyst gains some information about the key or plaintext. This information could be a few bits of the key, some information about the form of the plaintext, and so forth.

Unconditionally secure

- An algorithm is **unconditionally secure** if, no matter how much ciphertext a cryptanalyst has, there is not enough information to recover the plaintext.
- In point of fact, only a one-time pad (later on this) is unbreakable given infinite resources.
- All other cryptosystems are breakable in a ciphertext-only attack, simply by trying every possible key one by one and checking whether the resulting plaintext is meaningful. This is called a **brute-force** attack.

Computationally secure

- Cryptography is more concerned with cryptosystems that are **computationally infeasible to break**.
- An algorithm is considered **computationally secure** (sometimes called strong) if it cannot be broken with available resources, **either current or future**.
- Exactly what constitutes “available resources” is open to interpretation.

Complexity of Attacks

You can measure the complexity of an attack in different ways:

1. **Data complexity**. The amount of data needed as input to the attack.
2. **Processing complexity**. The time needed to perform the attack. This is often called the **work factor**.
3. **Storage requirements**. The amount of memory needed to do the attack.

Complexity of Attacks

- As a rule of thumb, the complexity of an attack is taken to be the **minimum** of the three factors.
- Some attacks involve trading off the three complexities: A faster attack might be possible at the expense of a greater storage requirement.
- Complexities are expressed as orders of magnitude. If an algorithm has a **processing complexity** of 2^{128}, then 2^{128} operations are required to break the algorithm.
- These operations may be complex and time-consuming.
- Still, if you assume that you have enough computing speed to perform a **million operations every second** and you set a **million parallel processors** against the task, it will still take over 10^{19} years to recover the key. **That's a billion times the age of the universe**.

Computing power Vs. Crypto

- While the complexity of an attack is constant (until some cryptanalyst finds a better attack, of course), computing power is not.
- There have been phenomenal advances in computing power during the last half-century and there is no reason to think this trend won't continue.
- Many cryptanalytic attacks are **perfect for parallel machines**:
  - The task can be broken down into billions of tiny pieces and none of the processors need to interact with each other.
- **Good cryptosystems are designed to be infeasible to break with the computing power that is expected to evolve many years in the future.**

Historical Terms

- Historically, a **code** refers to a cryptosystem that deals with linguistic units: words, phrases, sentences, and so forth.
- For example, the word “OCELOT” might be the ciphertext for the entire phrase “TURN LEFT 90 DEGREES,” the word “LOLLIPOP” might be the ciphertext for “TURN RIGHT 90 DEGREES,” and the words “BENT EAR” might be the ciphertext for “HOWITZER.” Codes of this type are not discussed here.
- Codes are only useful for **specialized circumstances**.
- Ciphers are useful for **any circumstance**. If your code has no entry for “ANTEATERS,” then you can’t say it. You can say anything with a cipher.

Steganography

- **Steganography** serves to hide secret messages in other messages, such that the secret's very existence is hidden.

- Generally the sender writes an innocuous message and then conceals a secret message on the same piece of paper.

- Historical tricks include invisible inks, tiny pin punctures on selected characters, minute differences between handwritten characters, pencil marks on typewritten characters, grilles which cover most of the message except for a few characters, and so on.

Outline

1.1 Terminology

1.2 Steganography

1.3 Substitution Ciphers and Transposition Ciphers

1.4 Simple XOR

1.5 One-Time Pads

1.6 Computer Algorithms

1.7 Large Numbers

Steganography

- More recently, people are hiding secret messages in **graphic images**.
- **Replace the least significant bit** of each byte of the image with the bits of the message.
- The graphical image won't change appreciably—most graphics standards specify more gradations of color than the human eye can notice—and the message can be stripped out at the receiving end.
- You can store a 64-kilobyte message in a 1024 × 1024 grey-scale picture this way. Several public-domain programs do this sort of thing.

Outline

1.1 Terminology

1.2 Steganography

1.3 Substitution Ciphers and Transposition Ciphers

1.4 Simple XOR

1.5 One-Time Pads

1.6 Computer Algorithms

1.7 Large Numbers

Ciphers

- Before computers, cryptography consisted of **character-based algorithms**. Different cryptographic algorithms either **substituted** characters for one another or **transposed** characters with one another.
- The better algorithms did both, many times each.
- Things are more complex these days, but the philosophy remains the same.
- The primary change is that algorithms work on bits instead of characters.
- This is actually just a change in the alphabet size: from 26 elements to two elements.
- Most good cryptographic algorithms still combine elements of substitution and transposition.

Substitution Ciphers

- **A substitution cipher** is one in which each character in the plaintext is substituted for another character in the ciphertext. The receiver inverts the substitution on the ciphertext to recover the plaintext.
- In classical cryptography, there are four types of substitution ciphers:
  - **A simple substitution cipher**, or **monoalphabetic cipher**, is one in which each character of the plaintext is replaced with a **corresponding character** of ciphertext. The cryptograms in newspapers are simple substitution ciphers.
  - **A homophonic substitution cipher** is like a simple substitution cryptosystem, except a single character of plaintext can **map to one of several characters** of ciphertext. For example, “A” could correspond to either 5, 13, 25, or 56, “B” could correspond to either 7, 19, 31, or 42, and so on.

Substitution Ciphers

- **A polygram substitution cipher** is one in which blocks of characters are encrypted in groups. For example, "ABA" could correspond to "RTQ," "ABB" could correspond to "SLL," and so on.
- **A polyalphabetic substitution cipher** is made up of multiple simple substitution ciphers. For example, there might be five different simple substitution ciphers used;
  - the particular one used changes with the position of each character of the plaintext.

Caesar Cipher

- This is a simple substitution Cipher
- In this famous Cipher, each plaintext character is replaced by the character three to the right modulo 26 (“A” is replaced by “D,” “B” is replaced by “E,”..., “W” is replaced by “Z,” “X” is replaced by “A,” “Y” is replaced by “B,” and “Z” is replaced by “C”).
- It’s actually simple, because the ciphertext alphabet is a rotation of the plaintext alphabet and not an arbitrary permutation.

ROT13

- ROT13 is a simple encryption program commonly found on UNIX systems; it is also a simple substitution cipher.
- In this cipher, “A” is replaced by “N,” “B” is replaced by “O,” and so on. Every letter is rotated 13 places.
- Encrypting a file twice with ROT13 restores the original file.
  - P = ROT13 (ROT13 (P))
- ROT13 is not intended for security; it is often used in Usenet posts to hide potentially offensive text, to avoid giving away the solution to a puzzle, and so forth.

Vigenère cipher

- In a Caesar cipher, each letter of the alphabet is shifted along some number of places; for example, shift 3, A would become D, B would become E and so on.
- **The Vigenère cipher consists of several Caesar ciphers in sequence with different shift values.**
- To encrypt, a table of alphabets can be used, termed a *tabula recta*, *Vigenère square*, or *Vigenère table*.
- It consists of the alphabet written out 26 times in different rows, **each alphabet shifted cyclically to the left** compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

Vigenère cipher

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X

Vigenère cipher

- For example, suppose that the plaintext to be encrypted is:
  - ATTACKATDAWN
- The person sending the message **chooses a keyword** and repeats it until it matches the length of the plaintext, for example, the keyword "LEMON":
  - LEMONLEMONLE
- The first letter of the plaintext, A, is enciphered using the alphabet in row L, which is the first letter of the key. This is done by looking at the letter in row L and column A of the Vigenère square, namely L. Similarly, for the second letter of the plaintext, the second letter of the key is used; the letter at row E and column T is X.
- Ciphertext: LXFOPVEFRNHR

Vigenère cipher

- Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in this row, and then using the column's label as the plaintext.

- For example, in row L (from *L*EMON), the ciphertext L appears in column A, which is the first plaintext letter.

- Next we go to row E (from L*E*MON), find the ciphertext X in column T, which is the second plaintext letter.

- **Who decrypts, should know the keyword LEMON**

Breaking substitution ciphers

- Simple substitution ciphers can be easily broken because the cipher does not hide the **underlying frequencies** of the different letters of the plaintext.

- Many algorithms solve these sorts of ciphers.

Breaking Homophonic SC

- Homophonic substitution ciphers were used as early as 1401 by the Duchy of Mantua.
- They are much more complicated to break than simple substitution ciphers, but still do not obscure all of the statistical properties of the plaintext language.
- With a known-plaintext attack, the ciphers are trivial to break.
- A ciphertext-only attack is harder, but only takes a few seconds on a computer.

Period of the cipher

- Polyalphabetic substitution ciphers have multiple one-letter keys, each of which is used to encrypt one letter of the plaintext.
- The first key encrypts the first letter of the plaintext, the second key encrypts the second letter of the plaintext, and so on. After all the keys are used, the keys are recycled.
- If there were 20 one-letter keys, then every twentieth letter would be encrypted with the same key. This is called the **period** of the cipher.
- In classical cryptography, ciphers with longer periods were significantly harder to break than ciphers with short periods.
- There are computer techniques that can easily break substitution ciphers with very long periods.

Running-key cipher

- A **running-key cipher** — sometimes called a book cipher — in which one text is used to encrypt another text, is another example of this sort of cipher.
- Even though this cipher has a period the length of the text, it can also be broken easily.

Transposition Ciphers

- In a **transposition cipher** the plaintext remains the same, but the order of characters is shuffled around.

- In a **simple columnar transposition cipher**, the plaintext is written horizontally onto a piece of graph paper of fixed width and the ciphertext is read off vertically.

- Decryption is a matter of writing the ciphertext vertically onto a piece of graph paper of identical width and then reading the plaintext off horizontally.

Transposition Ciphers

- Since the letters of the ciphertext are the same as those of the plaintext, a frequency analysis on the ciphertext would reveal that each letter has approximately the same likelihood as in English.
- This gives a very good clue to a cryptanalyst, who can then use a variety of techniques to determine the right ordering of the letters to obtain the plaintext.
- Putting the ciphertext through a second transposition cipher greatly enhances security.
- There are even more complicated transposition ciphers, but computers can break almost all of them.

Transposition Ciphers

- The German ADFGVX cipher, used during World War I, is a transposition cipher combined with a simple substitution. It was a very complex algorithm for its day but was broken by Georges Painvin, a French cryptanalyst.

- Although many modern algorithms use transposition, it is troublesome because it requires a lot of memory and sometimes requires messages to be only certain lengths.

- Substitution is far more common.

Rotor Machines

- In the 1920s, various mechanical encryption devices were invented to automate the process of encryption. Most were based on the concept of a **rotor**, a mechanical wheel wired to perform a **general substitution**.
- A **rotor machine** has a keyboard and a **series of rotors**, and implements a version of the Vigenère cipher.
- Each rotor is an arbitrary permutation of the alphabet, has 26 positions, and performs a simple substitution.
- For example, a rotor might be wired to substitute “F” for “A,” “U” for “B,” “L” for “C,” and so on.
- **And the output pins of one rotor are connected to the input pins of the next.**

Rotor Machines

- For example, in a 4-rotor machine the first rotor might substitute “F” for “A,” the second might substitute “Y” for “F,” the third might substitute “E” for “Y,” and the fourth might substitute “C” for “E”; **“C” would be the output ciphertext.**
- Then some of the rotors shift, so next time the substitutions will be different.
- It is the combination of several rotors and the gears moving them that makes the machine secure. Because the rotors all move at different rates, **the period for an n-rotor machine is 26^n**.
- Some rotor machines can also have a different number of positions on each rotor, further frustrating cryptanalysis. ☺

Rotor Machines

- The best-known rotor device is the Enigma. The Enigma was used by the Germans during World War II. The idea was invented by Arthur Scherbius and Arvid Gerhard Damm in Europe. It was patented in the United States by Arthur Scherbius [1383].
- The German Enigma had three rotors, chosen from a set of five, a plugboard that slightly permuted the plaintext, and a reflecting rotor that caused each rotor to operate on each plaintext letter twice.
- As complicated as the Enigma was, it was broken during World War II. First, a team of Polish cryptographers broke the German Enigma and explained their attack to the British.
- The Germans modified their Enigma as the war progressed, and the British continued to cryptanalyze the new versions. For explanations of how rotor ciphers work and how they were broken,

Outline

1.1 Terminology

1.2 Steganography

1.3 Substitution Ciphers and Transposition Ciphers

1.4 Simple XOR

1.5 One-Time Pads

1.6 Computer Algorithms

1.7 Large Numbers

Simple XOR

- **XOR** is exclusive-or operation: '^' in C or (+) in mathematical notation. It's a standard operation on bits:
  - 0 (+) 0 = 0
  - 0 (+) 1 = 1
  - 1 (+) 0 = 1
  - 1 (+) 1 = 0
- Also note that:
  - a (+) a = 0
  - a (+) b (+) b = a

Simple XOR

- The simple-XOR algorithm is really an embarrassment; it's nothing more than a Vigenère polyalphabetic cipher.
- It's here only because of its prevalence in commercial software packages, at least those in the MS-DOS and Macintosh worlds.
- Unfortunately, if a software security program proclaims that it has a "proprietary" encryption algorithm—significantly faster than DES—the odds are that it is some variant of this.

Simple XOR

- This is a symmetric algorithm. The plaintext is being XORed with a keyword to generate the ciphertext.
- Since XORing the same value twice restores the original, encryption and decryption use exactly the same program:
  - P (+) $K = C$
  - C (+) $K = P$
- There's no real security here. This kind of encryption is trivial to break, even without computers.
- It will only take a few seconds with a computer.

Outline

1.1 Terminology

1.2 Steganography

1.3 Substitution Ciphers and Transposition Ciphers

1.4 Simple XOR

1.5 One-Time Pads

1.6 Computer Algorithms

1.7 Large Numbers

One-Time Pads

- Believe it or not, there is a perfect encryption scheme. It's called a **one-time pad**, and was invented in 1917 by Major Joseph Mauborgne and AT&T's Gilbert Vernam.
- In cryptography, the **one-time pad (OTP)** is a type of encryption, which has been proven to be impossible to crack if used correctly.
- The plaintext is encrypted with a substitution cipher using a **secret random key** (or *pad*) as long as the plaintext, resulting in a **ciphertext of random data**.
- If the key is:
  - **truly random**
  - **as large as the plaintext**
  - **never reused in whole or part**
  - **kept secret**

  the ciphertext will be impossible to decrypt or break without knowing the key.

One-Time Pads

- Each key letter is used **exactly once**, for only one message.
- The sender encrypts the message and then **destroys** the used pages of the pad or used section of the tape.
- The receiver has an identical pad and uses each key on the pad, in turn, to decrypt each letter of the ciphertext.
- The receiver destroys the same pad pages or tape section after decrypting the message.
- New message—new key letters. For example, if the message is:
- ONETIMEPAD and the key sequence from the pad is
- TBFRGFARFM then the ciphertext is
- IPKLPSFHGQ because
  - O + T mod 26 = I
  - N + B mod 26 = P
  - E + F mod 26 = K

One-Time Pads

- Assuming an eavesdropper can't get access to the one-time pad used to encrypt the message, this scheme is perfectly secure.
- **A given ciphertext message is equally likely to correspond to any possible plaintext message of equal size.**
- Since every key sequence is equally likely (remember, the key letters are generated randomly), an adversary has no information with which to cryptanalyze the ciphertext.
- The key sequence could just as likely be:

  POYYAEAAZX which would decrypt to:

  SALMONEGGS or

  BXFGBMTMXM which would decrypt to:

  GREENFLUID

The power of OTP

- Since every plaintext message is equally possible, there is no way for the cryptanalyst to determine which plaintext message is the correct one.
- A random key sequence added to a nonrandom plaintext message produces a **completely random ciphertext message** and no amount of computing power can change that.
- Moreover: It has also been proven that any cipher with the **perfect secrecy property** must use keys with effectively the same requirements as OTP keys

- Claude Shannon proved, using information theory considerations, that the one-time pad has a property he termed ***perfect secrecy*****;** that is, the ciphertext *C* gives absolutely no additional information about the plaintext.

The power of OTP

- One-time pads have applications in today's world, primarily for **ultra-secure low-bandwidth channels**.
- The hotline between the United States and the former Soviet Union was rumored to be encrypted with a one-time pad. Many Soviet spy messages to agents were encrypted using one-time pads.
- **These messages are still secure today and will remain that way forever.** It doesn't matter how long the supercomputers work on the problem. Even after the aliens from Andromeda land with their massive spaceships and undreamed-of computing power, they will not be able to read the Soviet spy messages encrypted with one-time pads (unless they can also go back in time and get the one-time pads).

Outline

1.1 Terminology

1.2 Steganography

1.3 Substitution Ciphers and Transposition Ciphers

1.4 Simple XOR

1.5 One-Time Pads

1.6 Computer Algorithms

1.7 Large Numbers

Computer Algorithms

- There are many cryptographic algorithms. These are three of the most common:
  - **DES** (Data Encryption Standard) is the most popular computer encryption algorithm. DES is a U.S. and international standard. It is a symmetric algorithm; the same key is used for encryption and decryption.
  - **RSA** (named for its creators—Rivest, Shamir, and Adleman) is the most popular public-key algorithm. It can be used for both encryption and digital signatures.
  - **DSA** (Digital Signature Algorithm, used as part of the Digital Signature Standard) is another public-key algorithm. It cannot be used for encryption, but only for digital signatures.

Outline

1.1 Terminology

1.2 Steganography

1.3 Substitution Ciphers and Transposition Ciphers

1.4 Simple XOR

1.5 One-Time Pads

1.6 Computer Algorithms

1.7 Large Numbers

Large numbers

- Large numbers are used to describe different things in cryptography.

Large Numbers

Odds of being killed by lightning (per day)	1 in 9 billion (2^{33})
Odds of winning the top prize in a U.S. state lottery	1 in 4,000,000 (2^{22})
Odds of winning the top prize in a U.S. state lottery and being killed by lightning in the same day	1 in 2^{55}
Odds of drowning (in the U.S. per year)	1 in 59,000 (2^{16})
Odds of being killed in an automobile accident(in the U.S. in 1993)	1 in 6100 (2^{13})
Odds of being killed in an automobile accident(in the U.S. per lifetime)	1 in 88 (2^{7})
Time until the next ice age	14,000 (2^{14}) years
Time until the sun goes nova	10^{9} (2^{30}) years
Age of the planet	10^{9} (2^{30}) years
Age of the Universe	10^{10} (2^{34}) years
Number of atoms in the planet	10^{51} (2^{170})
Number of atoms in the sun	10^{57} (2^{190})
Number of atoms in the galaxy	10^{67} (2^{223})
Number of atoms in the Universe (dark matter excluded)	10^{77} (2^{265})
Volume of the Universe	10^{84} (2^{280}) cm^3

Large Numbers

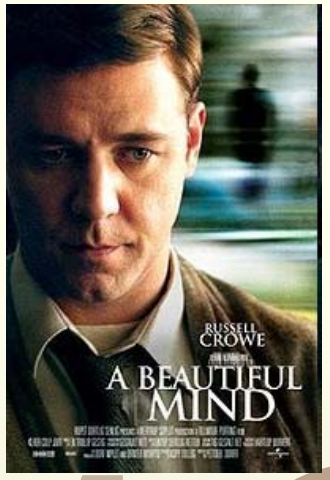
If the Universe is Closed:	
Total lifetime of the Universe	10^{11} (2^{37}) years
	10^{18} (2^{61}) seconds
If the Universe is Open:	
Time until low-mass stars cool off	10^{14} (2^{47}) years
Time until planets detach from stars	10^{15} (2^{50}) years
Time until stars detach from galaxies	10^{19} (2^{64}) years
Time until orbits decay by gravitational radiation	10^{20} (2^{67}) years
Time until black holes decay by the Hawking process	10^{64} (2^{213}) years
Time until all matter is liquid at zero temperature	10^{65} (2^{216}) years
Time until all matter decays to iron	10^{1026} years
Time until all matter collapses to black holes	10^{1076} years

End of Section

- Readings
  - Applied Cryptography. Chapter 1.

Break

- A Beautiful Mind is a 2001 American film based on the life of John Forbes Nash, Jr., a Nobel Laureate in Economics.
- After the conclusion of Nash's studies as a student at Princeton, he accepts a prestigious appointment at the Massachusetts Institute of Technology (MIT), along with his friends Sol and Bender.
- On a return visit to Princeton, Nash is invited to a secret Department of Defense facility in the Pentagon to **crack a complex encryption** of an enemy telecommunication. Nash is able to decipher the code mentally, to the astonishment of other codebreakers. Here, he encounters the mysterious William Parcher (Ed Harris), who belongs to the United States Department of Defense.



0011

Cryptographic Protocols

12

Outline

Protocol Building Blocks

0011

2.1 **Introduction to Protocols**

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

Definition

- A **protocol** is a series of steps, involving two or more parties, designed to accomplish a task.
- This is an important definition. A “series of steps” means that the protocol has a sequence, from start to finish.
- Every step must be executed in turn, and no step can be taken before the previous step is finished.
- “Involving two or more parties” means that at least two people are required to complete the protocol; one person alone does not make a protocol. A person alone can perform a series of steps to accomplish a task (like baking a cake), but this is not a protocol. (Someone else must eat the cake to make it a protocol.)
- Finally, “designed to accomplish a task” means that the protocol must achieve something. Something that looks like a protocol but does not accomplish a task is not a protocol—it’s a waste of time.

Characteristics

- Protocols have other characteristics as well:
  - Everyone involved in the protocol must know the protocol and all of the steps to follow **in advance**.
  - Everyone **involved** in the protocol must **agree** to follow it.
  - The protocol must be **unambiguous**; each step must be well defined and there must be no chance of a misunderstanding.
  - The protocol must be **complete**; there must be a specified action for every possible situation.

Cryptographic protocols

- A **cryptographic protocol** is a protocol that uses cryptography.
- The **parties can be friends** and trust each other implicitly or they can be **adversaries** and not trust one another to give the correct time of day.
- A cryptographic protocol involves some **cryptographic algorithm**, but generally the goal of the protocol is something beyond simple secrecy.
- The parties participating in the protocol might want to share parts of their secrets to compute a value, jointly generate a random sequence, convince one another of their identity, or simultaneously sign a contract.

Cryptographic protocols

- The whole point of using cryptography in a protocol is to prevent or detect eavesdropping and cheating.

- If you have never seen these protocols before, they will radically change your ideas of what mutually distrustful parties can accomplish over a computer network. In general, this can be stated as:
  - **It should not be possible to do more or learn more than what is specified in the protocol.**

Protocols: Failure and Success

- It is naïve to assume that **people** on computer networks are **honest**. It is naïve to assume that the **managers** of computer networks are **honest**. It is even naïve to assume that the **designers** of computer networks are **honest**. Most are, but the dishonest few can do a lot of damage.
- **By formalizing protocols, we can examine ways in which dishonest parties can subvert them**. Then we can develop protocols that are immune to that subversion.
- In some of the protocols it is possible for one of the participants to cheat the other.
- In others, it is possible for an eavesdropper to subvert the protocol or learn secret information.
- Some protocols fail because the designers weren't thorough enough in their requirements definitions. Others fail because their designers weren't thorough enough in their analysis.
- Like algorithms, it is much easier to prove insecurity than it is to prove security.

The Players

We will use these as our parties in the protocols:

Alice	First participant in all the protocols
Bob	Second participant in all the protocols
Carol	Participant in the three- and four-party protocols
Dave	Participant in the four-party protocols
Eve	Eavesdropper
Mallory	Malicious active attacker
Trent	Trusted arbitrator
Walter	Warden; he'll be guarding Alice and Bob in some protocols
Peggy	Prover
Victor	Verifier

Arbitrated Protocols

- An **arbitrator** is a disinterested third party trusted to complete a protocol.
- **Disinterested** means that the arbitrator has no vested interest in the protocol and no particular allegiance to any of the parties involved.
- **Trusted** means that all people involved in the protocol accept what he says as true, what he does as correct, and that he will complete his part of the protocol.
- **Arbitrators can help complete protocols between two mutually distrustful parties.**

Example of Problem

- For example, Alice is selling a car to Bob, a stranger.
- Bob wants to pay by check, but Alice has no way of knowing if the check is good.
- Alice wants the check to clear before she turns the title over to Bob.
- Bob, who doesn't trust Alice any more than she trusts him, doesn't want to hand over a check without receiving a title.

Example of Protocol

- Enter a lawyer trusted by both. With his help, Alice and Bob can use the following protocol to ensure that neither cheats the other:
  - **(1)** Alice gives the title to the lawyer.
  - **(2)** Bob gives the check to Alice.
  - **(3)** Alice deposits the check.
  - **(4)** After waiting a specified time period for the check to clear, the lawyer gives the title to Bob. If the check does not clear within the specified time period, Alice shows proof of this to the lawyer and the lawyer returns the title to Alice.

Adjudicated Protocols

- Because of the high cost of hiring arbitrators, arbitrated protocols can be subdivided into two lower-level **subprotocols**.
- One is a nonarbitrated subprotocol, executed every time parties want to complete the protocol.
- The other is an arbitrated subprotocol, executed only in exceptional circumstances—when there is a dispute. This special type of arbitrator is called an **adjudicator.**

Adjudicated Protocols

- Nonarbitrated subprotocol (executed every time):
  - **(1)** Alice and Bob negotiate the terms of the contract.
  - **(2)** Alice signs the contract.
  - **(3)** Bob signs the contract.
- Adjudicated subprotocol (executed only in case of a dispute):
  - **(4)** Alice and Bob appear before a judge.
  - **(5)** Alice presents her evidence.
  - **(6)** Bob presents his evidence.
  - **(7)** The judge rules on the evidence.

Self-Enforcing Protocols

- A **self-enforcing protocol** is the best type of protocol.
- The protocol itself guarantees fairness.
- **No arbitrator** is required to complete the protocol. **No adjudicator** is required to resolve disputes.
- The protocol is constructed so that there cannot be any disputes.
  - If one of the parties tries to cheat, the other party immediately detects the cheating and the protocol stops.
- Whatever the cheating party hoped would happen by cheating, doesn't happen.

Attacks against Protocols

- Cryptographic attacks can be directed against
  - the cryptographic algorithms used in protocols,
  - the cryptographic techniques used to implement the algorithms and protocols, or
  - the **protocols themselves**.
- Since this section of the book discusses protocols, we will assume that the cryptographic algorithms and techniques are secure.
- We will only examine attacks against the protocols.

Passive attack

- People can try various ways to attack a protocol.
- Someone not involved in the protocol can **eavesdrop** on some or all of the protocol.
- This is called a **passive attack**, because the attacker does not affect the protocol.
- All he can do is observe the protocol and attempt to gain information.
- This kind of attack corresponds to a ciphertext-only attack.
- Since passive attacks are difficult to detect, protocols try to prevent passive attacks rather than detect them.

Active Attack

- Alternatively, an attacker could try to alter the protocol to his own advantage.
- He could:
  - pretend to be someone else,
  - introduce new messages in the protocol,
  - delete existing messages,
  - substitute one message for another,
  - replay old messages,
  - interrupt a communications channel, or
  - alter stored information in a computer.
- These are called **active attacks**, because they require active intervention. The form of these attacks depends on the network.

Cheaters

- It is also possible that the attacker could be one of the parties involved in the protocol.
  - He may lie during the protocol or not follow the protocol at all. This type of attacker is called a **cheater**.
- **Passive cheaters** follow the protocol, but try to obtain more information than the protocol intends them to.
- **Active cheaters** disrupt the protocol in progress in an attempt to cheat.
- It is very difficult to maintain a protocol's security if most of the parties involved are **active cheaters**, but sometimes it is possible for legitimate parties to detect that active cheating is going on.
- Certainly, protocols should be secure against passive cheating.

Outline

Protocol Building Blocks

0011

2.1 Introduction to Protocols

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

Communications Using Symmetric Cryptography

- How do two parties communicate securely? They encrypt their communications, of course. The complete protocol is more complicated than that. Let's look at what must happen for Alice to send an encrypted message to Bob.
  - **(1)** Alice and Bob agree on a cryptosystem.
  - **(2)** Alice and Bob agree on a key.
  - **(3)** Alice takes her plaintext message and encrypts it using the encryption algorithm and the key. This creates a ciphertext message.
  - **(4)** Alice sends the ciphertext message to Bob.
  - **(5)** Bob decrypts the ciphertext message with the same algorithm and key and reads it.

Protocol Attack: 1

- What can Eve, sitting between Alice and Bob, learn from listening on this protocol?
- If all she hears is the transmission in step (4), she must try to cryptanalyze the ciphertext.
  - This passive attack is a ciphertext-only attack; we have algorithms that are resistant (as far as we know) to whatever computing power Eve could realistically bring to bear on the problem.
- Eve isn't stupid, though. She also wants to listen in on steps (1) and (2).
  - Then, she would know the algorithm and the key—just as well as Bob. When the message comes across the communications channel in step (4), all she has to do is decrypt it herself.

Knowledge of the key

- A good cryptosystem is one in which all the security is inherent in **knowledge of the key** and none is inherent in knowledge of the algorithm.
- This is why **key management** is so important in cryptography.
- With a symmetric algorithm, Alice and Bob can perform step (1) in public, but they must perform step (2) in secret.
- The **key must remain secret** before, during, and after the protocol—as long as the message must remain secret—otherwise the message will no longer be secure.

Protocol Attack: 2

- Mallory, an active attacker, could do a few other things.
- He could attempt to break the communications path in step (4), ensuring that Alice could not talk to Bob at all.
- Mallory could also intercept Alice's messages and substitute his own.
- If he knew the key (by intercepting the communication in step (2), or by breaking the cryptosystem), he could encrypt his own message and send it to Bob in place of the intercepted message.
- Bob would have no way of knowing that the message had not come from Alice.
- If Mallory didn't know the key, he could only create a replacement message that would decrypt to strange meaning.
  - Bob, thinking the message came from Alice, might conclude that either the network or Alice had some serious problems. ☺

Protocol Attack: Trust

- What about Alice? What can she do to disrupt the protocol? She can give a copy of the key to Eve.
- Now Eve can read whatever Bob says. She can reprint his words in *The New York Times*.
- **Although serious, this is not a problem with the protocol.**
- There is nothing to stop Alice from giving Eve a copy of the plaintext at any point during the protocol.
- Of course, Bob could also do anything that Alice could. **This protocol assumes that Alice and Bob trust each other.**

Problems of symmetric cryptosystems

- **Keys must be distributed in secret.** They are as valuable as all the messages they encrypt, since knowledge of the key gives knowledge of all the messages. For encryption systems that span the world, this can be a daunting task. Often couriers hand-carry keys to their destinations.
- If a key is **compromised** (stolen, guessed, extorted, bribed, etc.), then Eve can decrypt all message traffic encrypted with that key. She can also pretend to be one of the parties and produce false messages to fool the other party.
- Assuming a separate key is used for each pair of users in a network, the total number of keys increases rapidly as the number of users increases. A network of n users requires $n(n - 1)/2$ keys. For example, 10 users require 45 different keys to talk with one another and 100 users require 4950 keys. This problem can be minimized by keeping the number of users small, but that is not always possible.

Outline

Protocol Building Blocks

0011

2.1 Introduction to Protocols

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

One-Way Functions

- The notion of a **one-way function** is central to public-key cryptography. While not protocols in themselves, one-way functions are a fundamental building block for most of the protocols discussed in this book.

- One-way functions are relatively easy to compute, but significantly harder to reverse. That is, given x it is easy to compute $f(x)$, but given $f(x)$ it is hard to compute x.

- In this context, "hard" is defined as something like: **It would take millions of years to compute x from $f(x)$, even if all the computers in the world were assigned to the problem.**

Trapdoor One-Way Functions

- A **trapdoor one-way function** is a special type of one-way function, one with a secret trapdoor.
- It is easy to compute in one direction and hard to compute in the other direction. But, if you know the secret, you can easily compute the function in the other direction.
- That is, it is easy to compute $f(x)$ given x, and hard to compute x given $f(x)$. However, there is some secret information, y, such that given $f(x)$ and y it is easy to compute x.

Outline

Protocol Building Blocks

0011

2.1 Introduction to Protocols

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

One-Way Hash Functions

- A **one-way hash function** has many names: compression function, contraction function, message digest, fingerprint, cryptographic checksum, message integrity check (MIC), and manipulation detection code (MDC).
- Whatever you call it, it is central to modern cryptography.
- One-way hash functions are another building block for many protocols.

Hash Functions

- Hash functions have been used in computer science for a long time.
- A hash function is a function, mathematical or otherwise, that takes a variable-length input string (called a **pre-image**) and converts it to a fixed-length (generally smaller) output string (called a **hash value**).
- A simple hash function would be a function that takes pre-image and returns a byte consisting of the XOR of all the input bytes (this is not one way)

One-Way Hash Functions

- A one-way hash function is a hash function that works in one direction: It is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value.
- A good one-way hash function is also **collision-free**: It is hard to generate two pre-images with the same hash value.
- The hash function is **public**; there's no secrecy to the process.
  - The security of a one-way hash function is its **one-wayness**. The output is not dependent on the input in any discernible way.
- A single bit change in the pre-image changes, on the average, half of the bits in the hash value. Given a hash value, it is **computationally unfeasible** to find a pre-image that hashes to that value.

Message Authentication Codes

- A **message authentication code** (MAC), also known as a data authentication code (DAC), is a one-way hash function with the addition of a secret key.
- The hash value is a function of both the pre-image and the key.
- The theory is exactly the same as hash functions, except only someone with the key can verify the hash value. You can create a MAC out of a hash function or a block encryption algorithm; there are also dedicated MACs.

Outline

Protocol Building Blocks

0011

2.1 Introduction to Protocols

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

Communications Using Public-Key Cryptography

- In 1976, Whitfield Diffie and Martin Hellman changed the paradigm of cryptography forever.
- The NSA has claimed knowledge of the concept as early as 1966, but has offered no proof.
- They described **public-key cryptography**. They used two different keys—one public and the other private.
- It is **computationally hard** to deduce the private key from the public key.
- Anyone with the public key can encrypt a message but not decrypt it. Only the person with the private key can decrypt the message.

Public key

- Mathematically, the process is based on the trap-door one-way functions previously discussed.
- Encryption is the easy direction. Instructions for encryption are the public key; anyone can encrypt a message.
- Decryption is the hard direction.
- It's made hard enough that people with Cray computers and thousands (even millions) of years couldn't decrypt the message without the secret.
- The secret, or trapdoor, is the **private key**.

Public key

- This is how Alice can send a message to Bob using public-key cryptography:
  - **(1)** Alice and Bob agree on a public-key cryptosystem.
  - **(2)** Bob sends Alice his public key.
  - **(3)** Alice encrypts her message using Bob's public key and sends it to Bob.
  - **(4)** Bob decrypts Alice's message using his private key.

Public key Vs. Symmetric

- Notice how public-key cryptography solves the key-management problem with symmetric cryptosystems.
- Before, Alice and Bob had to agree on a key in secret. Alice could choose one at random, but she still had to get it to Bob.
- She could hand it to him sometime beforehand, but that requires foresight. She could send it to him by secure courier, but that takes time.
- Public-key cryptography makes it easy. With no prior arrangements, Alice can send a secure message to Bob. Eve, listening in on the entire exchange, has Bob's public key and a message encrypted in that key, but cannot recover either Bob's private key or the message.

Public key in Networks

- More commonly, a network of users agrees on a public-key cryptosystem. Every user has her own public key and private key, and the public keys are all published in a database somewhere. Now the protocol is even easier:
  - **(1)** Alice gets Bob's public key from the database.
  - **(2)** Alice encrypts her message using Bob's public key and sends it to Bob.
  - **(3)** Bob then decrypts Alice's message using his private key.

Hybrid Cryptosystems

- In the real world, public-key algorithms are not a substitute for symmetric algorithms. They are not used to encrypt messages; they are used to encrypt keys. There are two reasons for this:
- **Public-key algorithms are slow.**
  - Symmetric algorithms are generally at least 1000 times faster than public-key algorithms. Yes, computers are getting faster and faster, and in 15 years computers will be able to do public-key cryptography at speeds comparable to symmetric cryptography today.
  - But bandwidth requirements are also increasing, and there will always be the need to **encrypt data faster than public-key cryptography can manage**.

Hybrid Cryptosystems

- Public-key cryptosystems are **vulnerable** to chosen-plaintext attacks.

  If $C = E(P)$, when P is one plaintext out of a set of n possible plaintexts, then a cryptanalyst only has to encrypt all n possible plaintexts and compare the results with C (remember, the encryption key is public).
- He won't be able to recover the decryption key this way, but he will be able to determine P.

Hybrid Cryptosystems

- In most practical implementations public-key cryptography is used to secure and distribute **session keys**; those session keys are used with symmetric algorithms to secure message traffic.
- This is sometimes called a **hybrid cryptosystem**.
  - **(1)** Bob sends Alice his public key.
  - **(2)** Alice generates a random session key, K, encrypts it using Bob's public key, and sends it to Bob.

    $E_B(K)$
  - **(3)** Bob decrypts Alice's message using his private key to recover the session key.

    $D_B(E_B(K)) = K$
  - **(4)** Both of them encrypt their communications using the same session key.

Outline

Protocol Building Blocks

0011

2.1 Introduction to Protocols

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

Digital Signatures

Handwritten signatures have long been used as proof of authorship:

1. The signature is authentic. The signature convinces the document's recipient that the signer deliberately signed the document.
2. The signature is unforgeable. The signature is proof that the signer, and no one else, deliberately signed the document.
3. The signature is not reusable. The signature is part of the document; an unscrupulous person cannot move the signature to a different document.
4. The signed document is unalterable. After the document is signed, it cannot be altered.
5. The signature cannot be repudiated. The signature and the document are physical things. The signer cannot later claim that he or she didn't sign it.

In reality, none of these statements about signatures is completely true.

Signing Documents with Symmetric Cryptosystems and an Arbitrator

- Alice wants to sign a digital message and send it to Bob. With the help of Trent and a symmetric cryptosystem, she can.
- Trent is a powerful, trusted arbitrator.
- He can communicate with both Alice and Bob (and everyone else who may want to sign a digital document).
- He shares a secret key, K_A, with Alice, and a different secret key, K_B, with Bob.
- These keys have been established long before the protocol begins and can be reused multiple times for multiple signings.

Signing Documents with Symmetric Cryptosystems and an Arbitrator

(1) Alice encrypts her message to Bob with K_A and sends it to Trent.

(2) Trent decrypts the message with K_A.

(3) Trent takes the decrypted message and a statement that he has received this message from Alice, and encrypts the whole bundle with K_B.

(4) Trent sends the encrypted bundle to Bob.

(5) Bob decrypts the bundle with K_B. He can now read both the message and Trent's certification that Alice sent it.

Signing Documents with Symmetric Cryptosystems and an Arbitrator

1. This signature is authentic. Trent is a trusted arbitrator and Trent knows that the message came from Alice.

2. This signature is unforgeable. Only Alice (and Trent, but everyone trusts him) knows K_A, so only Alice could have sent Trent a message encrypted with K_A.

3. This signature is not reusable. If Bob tried to take Trent's certification and attach it to another message, an arbitrator (it could be Trent or it could be a completely different arbitrator) would ask Bob to produce both the message and Alice's encrypted message. The arbitrator would then encrypt the message with K_A and see that it did not match the encrypted message that Bob gave him.

Signing Documents with Symmetric Cryptosystems and an Arbitrator

4. The signed document is unalterable. Were Bob to try to alter the document after receipt, Trent could prove foul play in exactly the same manner just described.

5. The signature cannot be repudiated. Even if Alice later claims that she never sent the message, Trent's certification says otherwise. Remember, Trent is trusted by everyone; what he says is true.

Signing Documents with Symmetric Cryptosystems and an Arbitrator

- These protocols work, but they're time-consuming for Trent.
- He must **spend his days decrypting and encrypting messages**, acting as the intermediary between every pair of people who want to send signed documents to one another.
- He must **keep a database of messages** (although this can be avoided by sending the recipient a copy of the sender's encrypted message).
- He is a bottleneck in any communications system, even if he's a mindless software program.
- **Harder still is creating and maintaining someone like Trent**, someone that everyone on the network trusts.
- **Trent has to be infallible;** if he makes even one mistake in a million signatures, no one is going to trust him.
- **Trent has to be completely secure.** If his database of secret keys ever got out or if someone managed to modify his programming, everyone's signatures would be completely useless. False documents purported to be signed years ago could appear.

Signing Documents with Public-Key Cryptography

- There are public-key algorithms that can be used for digital signatures.
- In some algorithms—RSA is an example—either the public key or the private key can be used for encryption.
- Encrypt a document using your private key, and you have a secure digital signature.
- In other cases—DSA is an example —there is a separate algorithm for digital signatures that cannot be used for encryption. This idea was first invented by Diffie and Hellman.
- The basic protocol is simple:
  - **(1)** Alice encrypts the document with her private key, thereby signing the document.
  - **(2)** Alice sends the signed document to Bob.
  - **(3)** Bob decrypts the document with Alice's public key, thereby verifying the signature.

Signing Documents and Timestamps

- Actually, Bob can cheat Alice in certain circumstances. He can reuse the document and signature together. This is no problem if Alice signed a contract (what's another copy of the same contract, more or less?), but it can be very exciting if Alice signed a digital check.
- Let's say Alice sends Bob a signed digital check for $100. Bob takes the check to the bank, which verifies the signature and moves the money from one account to the other.
- Bob, who is an unscrupulous character, saves a copy of the digital check. The following week, he again takes it to the bank (or maybe to a different bank). The bank verifies the signature and moves the money from one account to the other. If Alice never balances her checkbook, Bob can keep this up for years.

Signing Documents and Timestamps

- Consequently, digital signatures often include timestamps.
- The date and time of the signature are attached to the message and signed along with the rest of the message.
- The bank stores this timestamp in a database. Now, when Bob tries to cash Alice's check a second time, the bank checks the timestamp against its database.
- Since the bank already cashed a check from Alice with the same timestamp, the bank calls the police.
- Bob then spends 15 years in Leavenworth prison reading up on cryptographic protocols. ☺

Signing Documents with Public-Key Cryptography and One-Way Hash Functions

- In practical implementations, public-key algorithms are often too inefficient to sign long documents. To save time, digital signature protocols are often implemented with one-way hash functions.
- Instead of signing a document, Alice **signs the hash of the document**. In this protocol, both the one-way hash function and the digital signature algorithm are agreed upon beforehand.

  (1) Alice produces a one-way hash of a document.

  (2) Alice encrypts the hash with her private key, thereby signing the document.

  (3) Alice sends the document and the signed hash to Bob.

  (4) Bob produces a one-way hash of the document that Alice sent. He then, using the digital signature algorithm, decrypts the signed hash with Alice's public key. If the signed hash matches the hash he generated, the signature is valid.

Signing Documents with Public-Key Cryptography and One-Way Hash Functions

- Speed increases drastically and, since the chances of two different documents having the same 160-bit hash are only one in 2^{160}, anyone can safely equate a signature of the hash with a signature of the document.
- If a non-one-way hash function were used, it would be an easy matter to create multiple documents that hashed to the same value.

Signing Documents with Public-Key Cryptography and One-Way Hash Functions

- This protocol has other benefits.
- First, the signature can be kept **separate** from the document.
- Second, the recipient's storage requirements for the document and signature are **much smaller**.
- An archival system can use this type of protocol to verify the existence of documents without storing their contents.
- The central database could just **store the hashes of files**. It doesn't have to see the files at all; users submit their hashes to the database, and the database timestamps the submissions and stores them.
- If there is any disagreement in the future about who created a document and when, the database could resolve it by finding the hash in its files.

Algorithms and Terminology

- There are many digital signature algorithms. All of them are public-key algorithms with secret information to sign documents and public information to verify signatures.
- Sometimes the signing process is called **encrypting with a private key** and the verification process is called **decrypting with a public key**.
  - This is misleading and is only true for one algorithm, RSA.
- And different algorithms have different implementations.
  - For example, one-way hash functions and timestamps sometimes add extra steps to the process of signing and verifying. Many algorithms can be used for digital signatures, but not for encryption.

Algorithms and Terminology

- In general, we will refer to the signing and verifying processes without any details of the algorithms involved. Signing a message with private key K is:

  $S_K(M)$

- and verifying a signature with the corresponding public key is:

  $V_K(M)$

- The bit string attached to the document when signed (in the previous example, the one-way hash of the document encrypted with the private key) will be called the **digital signature**, or just the **signature**.
- The entire protocol, by which the receiver of a message is convinced of the identity of the sender and the integrity of the message, is called **authentication**.

Multiple Signatures

- How could Alice and Bob sign the same digital document?
- Without one-way hash functions, there are two options.
- One is that Alice and Bob **sign separate copies** of the document itself. The resultant message would be over twice the size of the original document.
- The second is that Alice signs the document first and then **Bob signs Alice's signature**.
- This works, but it is impossible to verify Alice's signature without also verifying Bob's.

Multiple Signatures

- With one-way hash functions, multiple signatures are easy:
  - **(1)** Alice signs the hash of the document.
  - **(2)** Bob signs the hash of the document.
  - **(3)** Bob sends his signature to Alice.
  - **(4)** Alice sends the document, her signature, and Bob's signature to Carol.
  - **(5)** Carol verifies both Alice's signature and Bob's signature.
- Alice and Bob can do steps (1) and (2) either in parallel or in series. In step (5), Carol can verify one signature without having to verify the other.

Outline

Protocol Building Blocks

0011

2.1 Introduction to Protocols

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

Applications of Digital Signatures

- One of the earliest proposed applications of digital signatures was to facilitate the **verification of nuclear test ban treaties**.
- The United States and the Soviet Union permitted each other to put seismometers on the other's soil to monitor nuclear tests.
- The problem was that each country needed to assure itself that the host nation was not tampering with the data from the monitoring nation's seismometers. Simultaneously, the host nation needed to assure itself that the monitor was sending only the specific information needed for monitoring.
- Conventional authentication techniques can solve the first problem, **but only digital signatures can solve both problems**.
- The host nation can read, but not alter, data from the seismometer, and the monitoring nation knows that the data has not been tampered with.

Digital Signatures with Encryption

- By combining digital signatures with public-key cryptography, we develop a protocol that combines the security of encryption with the authenticity of digital signatures. Think of a letter from your mother: The signature provides proof of authorship and the envelope provides privacy.
  - **(1)** Alice signs the message with her private key.

    $S_A(M)$
  - **(2)** Alice encrypts the signed message with Bob's public key and sends it to Bob.

    $E_B(S_A(M))$
  - **(3)** Bob decrypts the message with his private key.

    $D_B(E_B(S_A(M))) = S_A(M)$
  - **(4)** Bob verifies with Alice's public key and recovers the message.

    $V_A(S_A(M)) = M$

Digital Signatures with Encryption

- There's no reason Alice has to use the same public-key/private-key key pair for encrypting and signing.
- She can have two key pairs: one for encryption and the other for signatures.
- Separation has its advantages:
  - she can surrender her encryption key to the police without compromising her signature,
  - one key can be escrowed without affecting the other
  - the keys can have different sizes and can expire at different times.
- Of course, timestamps should be used with this protocol to prevent reuse of messages. Timestamps can also protect against other potential pitfalls, such as the one described below.

Resending the Message as a Receipt

- Consider an implementation of this protocol, with the additional feature of confirmation messages. Whenever Bob receives a message, he returns it as a **confirmation of receipt**.

  (1) Alice signs a message with her private key, encrypts it with Bob's public key, and sends it to Bob.

  $E_B(S_A(M))$

  (2) Bob decrypts the message with his private key and verifies the signature with Alice's public key, thereby verifying that Alice signed the message and recovering the message.

  $V_A(D_B(E_B(S_A(M)))) = M$

  (3) Bob signs the message with his private key, encrypts it with Alice's public key, and sends it back to Alice.

  $E_A(S_B(M))$

  (4) Alice decrypts the message with her private key and verifies the signature with Bob's public key. If the resultant message is the same one she sent to Bob, she knows that Bob received the message accurately.

Attacks against Public-Key Cryptography

- In all these public-key cryptography protocols, we glossed over how Alice gets Bob's public key.
- The easiest way to get someone's public key is from a **secure database** somewhere. The database has to be public, so that anyone can get anyone else's public key.
- The database also has to be **protected** from write-access by anyone except Trent; otherwise Mallory could substitute any public key for Bob's.
- After he did that, Bob couldn't read messages addressed to him, but Mallory could.

Attacks against Public-Key Cryptography

- Even if the public keys are stored in a secure database, Mallory could still **substitute** one for another during transmission.
- To prevent this, Trent can sign each public key with his own private key. Trent, when used in this manner, is often known as a **Key Certification Authority** or **Key Distribution Center** (**KDC**).
- In practical implementations, the KDC signs a compound message consisting of the user's **name**, his public **key**, and any other important **information** about the user.
- This signed compound message is stored in the KDC's database.
- When Alice gets Bob's key, she verifies the KDC's signature to assure herself of the key's validity.

Outline

Protocol Building Blocks

0011

2.1 Introduction to Protocols

2.2 Communications Using Symmetric Cryptography

2.3 One-Way Functions

2.4 One-Way Hash Functions

2.5 Communications Using Public-Key Cryptography

2.6 Digital Signatures

2.7 Digital Signatures with Encryption

2.8 Random and Pseudo-Random-Sequence Generation

Random and Pseudo-Random-Sequence Generation

- Why even bother with random-number generation in a book on cryptography? There's already a random-number generator built into most every compiler, a mere function call away. Why not use that?
- Unfortunately, those random-number generators are almost definitely not secure enough for cryptography, and probably not even very random.
- Most of them are embarrassingly bad.

Random numbers and Computers

- Cryptography is **extremely sensitive** to the properties of random-number generators.
- The problem is that a random-number generator doesn't produce a random sequence.
  - It probably doesn't produce anything that looks even remotely like a random sequence.
- **Of course, it is impossible to produce something truly random on a computer**. Donald Knuth quotes John von Neumann as saying: "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin"

Random numbers and Computers

- Computers are **deterministic beasts**: Stuff goes in one end, completely predictable operations occur inside, and different stuff comes out the other end.
- Put the same stuff in on two separate occasions and the same stuff comes out both times. Put the same stuff into two identical computers, and the same stuff comes out of both of them.
- A computer can only be in a **finite number of states** (a large number, but a finite number nonetheless), and the stuff that comes out will always be a deterministic function of the stuff that went in and the computer's current state.
- That means that any random-number generator on a computer (at least, on a finite-state machine) is, by definition, **periodic**. Anything that is periodic is, by definition, **predictable**.
- **And if something is predictable, it can't be random.**
- **A true random-number generator requires some random input; a computer can't provide that.**

Pseudo-Random Sequences

- The best a computer can produce is a **pseudo-random-sequence generator**.
- A pseudo-random sequence is one that looks random.
- The sequence's period should be long enough so that a finite sequence of reasonable length—that is, one that is actually used — **is not periodic**. If you need a billion random bits, don't choose a sequence generator that repeats after only sixteen thousand bits.
- These relatively short nonperiodic subsequences should be as indistinguishable as possible from random sequences.

Pseudo-Random Sequences

- For our purposes, a sequence generator is pseudo-random if it has this property:

  1. **It looks random. This means that it passes all the statistical tests of randomness that we can find.**

- A lot of effort has gone into producing good pseudo-random sequences on computer. Discussions of generators abound in the academic literature, along with various tests of randomness. All of these generators are periodic (there's no escaping that); but with potential periods of 2^{256} bits and higher, they can be used for the largest applications.
- The problem is still those weird correlations and strange results. Every pseudo-random-sequence generator is going to produce them if you use them in a certain way.
- And that's what a cryptanalyst will use to attack the system.

Cryptographically Secure Pseudo-Random Sequences

- Cryptographic applications demand much more of a pseudo-random-sequence generator than do most other applications. Cryptographic randomness doesn't mean just statistical randomness, although that's part of it. For a sequence to be **cryptographically secure pseudo-random**, it must also have this property:
  2. **It is unpredictable**. It must be computationally infeasible to predict what the next random bit will be, given complete knowledge of the algorithm or hardware generating the sequence and all of the previous bits in the stream.

Cryptographically Secure Pseudo-Random Sequences

- Cryptographically secure pseudo-random sequences should not be compressible...unless you know the key. The key is generally the **seed** used to set the initial state of the generator.

- Like any cryptographic algorithm, cryptographically secure pseudo-random-sequence generators are subject to attack. Just as it is possible to break an encryption algorithm, it is possible to break a cryptographically secure pseudo-random-sequence generator.

- **Making generators resistant to attack is what cryptography is all about.**

Real Random Sequences

- Is there such a thing as randomness? What is a random sequence? How do you know if a sequence is random? Is "101110100" more random than "101010101"?
- Quantum mechanics tells us that there is honest-to-goodness randomness in the real world. But can we preserve that randomness in the deterministic world of computer chips and finite-state machines?
- Philosophy aside, from our point of view a sequence generator is **real random** if it has this additional third property:
  - **3. It cannot be reliably reproduced**. If you run the sequence generator twice with the exact same input (at least as exact as humanly possible), you will get two completely unrelated random sequences.

Real Random Sequences

- The output of a generator satisfying these three properties will be good enough for a one-time pad, key generation, and any other cryptographic applications that require a truly random sequence generator.

- The difficulty is in determining **whether a sequence is really random**. If I repeatedly encrypt a string with DES and a given key, I will get a nice, random-looking output;
  - you won't be able to tell that it's nonrandom unless you rent time on the NSA's DES cracker.

End of Lesson

- Readings
  - Chapter 1 and 2