Introduction to Computer Security

Cryptography II

Dr. Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: October 25, 2019 at 14:41

Why Cryptosystems Fail

Suggestion: Read "Why Cryptosystems Fail" by Ross Anderson, available on-line.

> *It turns out that the threat model commonly used by cryptosystem designers was wrong: most frauds were not caused by cryptanalysis or other technical attacks, but by implementation errors and management failures. This suggests that a paradigm shift is overdue in computer security.*
> *(from the abstract)*

Data Encryption Standard

In 1972, the NBS (subsequently NIST) issued a call for proposals to produce an encryption algorithm with the characteristics:

- able to provide a high level of security;
- specified and easy to understand;
- publishable (security doesn't depend on algorithm secrecy);
- available to all users;
- adaptable for use in diverse applications;
- economical to implement in electronic devices;
- efficient to use;
- able to be validated.
- exportable.

DES

The Data Encryption Standard (DES) was developed for the U.S. government for use by the general public. It is a cryptographic standard in wide use in the U.S. and abroad.

The NBS choice for DES was based on the *Lucifer* algorithm developed by IBM.

DES was adopted as a U.S. federal standard in November 1976, authorized by NBS for use on all public and private sector unclassified communication. Eventually, it was accepted as an international standard by the International Standards Organization.

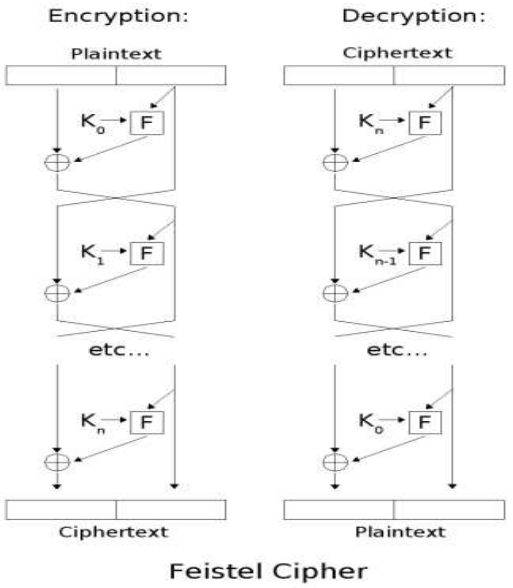
Feistel Ciphers

DES is an example of a *Feistel cipher*, a special class of iterated block ciphers where the ciphertext is calculated from the plaintext (and key) by repeated application (rounds) of the same transformation or function. Feistel ciphers are sometimes called *DES-like ciphers*.

The text being encrypted is split into two halves. The function f is applied to one half using a subkey and the output of f is XORed with the other half. The two halves are then swapped. Each round follows the same pattern except for the last round, where there is no swap.

In Feistel ciphers, encryption and decryption are structurally identical, though the subkeys used during encryption in each round are taken in reverse during decryption.

Feistel Ciphers



Feistel Cipher

Weak Keys

One of the goals of analyzing any encryption scheme is to find potential theoretical flaws or weaknesses in the algorithm. For example, in many block ciphers there are some keys that should be avoided because of reduced complexity of the cipher.

In all Feistel algorithms, from the original key you generate a separate subkey for each round of the algorithm. It is important to avoid keys that generate identical subkeys in more than one round.

DES has four *weak keys*: the same subkey is generated in every round. DES has twelve *semi-weak keys*: only two subkeys are generated and alternate in rounds.

Overview of the Algorithm

The DES algorithm is a combination of substitution and transposition. The two techniques are repeatedly applied through 16 cycles or rounds.

Plaintext is encrypted in 64-bit blocks. The key is 64-bits, but 8 bits are used as check digits and don't affect the encryption. Thus, the key is effectively 56-bits.

Key Schedule

1. Initially, 56 bits of the key are selected from the initial 64 by Permuted Choice 1 (PC-1).
2. The 56 bits are then divided into two 28-bit halves; each half is thereafter treated separately.
3. In successive rounds, both halves are rotated left by one or two bits (specified for each round), and then 48 subkey bits are selected by Permuted Choice 2 (PC-2), 24 bits from the left half, and 24 from the right.
4. For successive keys, the halves are rotated so that a different set of bits is used in each subkey; each bit is used in approximately 14 out of the 16 subkeys.
5. The key schedule for decryption is similar—it must generate the keys in the reverse order. Hence the rotations are to the right, rather than the left.

Scrambling

The scrambling function operates on half a block (32 bits) at a time and consists of four stages:

Expansion: the 32-bit half-block is expanded to 48 bits using the expansion permutation by duplicating some of the bits.

Key mixing: the result is combined with a subkey using an XOR operation. Sixteen 48-bit subkeys—one for each round— are derived from the main key using the key schedule.

Scrambling (continued)

Substitution: after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the S-boxes, or substitution boxes.

- Each of eight S-boxes replaces its six input bits with four output bits according to a non-linear transformation, provided in the form of a lookup table.
- The S-boxes provide the core of the security of DES—without them, the cipher would be linear, and trivially breakable.

Permutation: finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the P-box.

Characteristics

The algorithm uses only standard arithmetic and logical operations on 64-bit numbers, and hence can be implemented on standard hardware or on single-purpose chips.

Research has shown that almost any change to the algorithm weakens it. Reducing the iterations to 15 or changing the substitution or permutation strategy weakens the algorithm such that it can be broken.

The algorithm is its own inverse, if the keys are used in reverse order. (The keys here are the shifted permuted keys generated from k in each cycle.) That is:

$$E(k, E(k', P)) = P.$$

Security of the Algorithm

The major weakness of DES is that the algorithm uses a fixed 56-bit key. It would be extremely difficult to extend this without significant changes to the algorithm.

In 1977, it was infeasible to break DES exhaustively, by trying all 2^{56} (approximately 10^{15}) keys. In 1997, researchers using 3500 machines in parallel broke a DES encryption in around 4 months. In 1998, researchers built a "DES cracker" machine for around $100,000 that could do it in around four days.

There has long been a worry that the government left a "trapdoor" in the algorithm, though long study has not found any.

Modes of Usage: ECB

There are various modes of usage for DES. *The same general modes are used for all iterated block ciphers, including AES and others.*

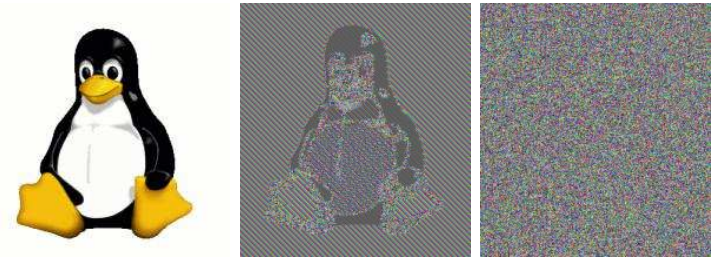
ECB: Electronic Code Book This means to encrypt each block independently, using the same key for each. It is rarely used.

ECB is called a *block encryption mode*. Because the text length may not be a multiple of the block size, you may have to pad the final block.

The problem with ECB is that identical blocks in the plaintext will yield identical blocks in the ciphertext. This is a problem for plaintext with frequent repeats, such as internet packet traffic.

The Problem with ECB

This is a very graphic illustration of the problem with ECB:



Original With ECB Another Mode

Modes of Usage: CBC

CBC: Cipher Block Chaining Means to XOR each successive plaintext block with the previous ciphertext block and then encrypt. An initialization vector c_0 is used as a "seed" for the process. CBC is also a *block encryption mode*. May have to pad the final block.

Blocks are effectively randomized before encryption, so that identical blocks in the plaintext yield different blocks in the ciphertext.

CBC (continued)

The process of encryption with CBC can be characterized algebraically as follows:

$$C_0 = IV$$

$$C_i = E_K(P_i \oplus C_{i-1})$$

where E_K is the block encryption algorithm with key K.

Notice: to encrypt a block with CBC you have to encrypt all of the previous blocks. This makes it not entirely suitable for, say, disk encryption. Why?

What is the corresponding decryption algorithm? To decrypt ciphertext block C_i do you have to decrypt all previous blocks?

CBC Vulnerabilities (Content Leaks)

Though better than ECB, CBC still has some weaknesses.

Observed changes: An attacker able to observe changes to ciphertext over time will be able to spot the first block that changed.

Content Leak: If an attacker can find two identical ciphertext blocks, C_i and C_j, he can derive the following relation:

$$C_{i-1} \oplus C_{j-1} = P_i \oplus P_j.$$

Thus, the attacker can compute $P_i \oplus P_j$ and thereby derive information about two plaintext blocks.

CBC Vulnerabilities (Watermark)

Watermarking Attack: Assume that the (known) IV associated with sector k is IV_k, and that the attacker can cause the system to encrypt and store two chosen plaintexts P_1 and P_2 as the first blocks in two sections, numbered i and j, respectively. Let's call these ciphertext blocks $C_{i,1}$ and $C_{j,1}$, respectively. By the definition of CBC mode,

$$C_{i,1} = E_K(P_1 \oplus IV_i).$$

The attacker chooses

$$P_2 = P_1 \oplus IV_i \oplus IV_j.$$

Then

$$C_{j,1} = E_K(P_2 \oplus IV_j) = E_K(P_1 \oplus IV_i \oplus IV_j \oplus IV_j) = C_{i,1}.$$

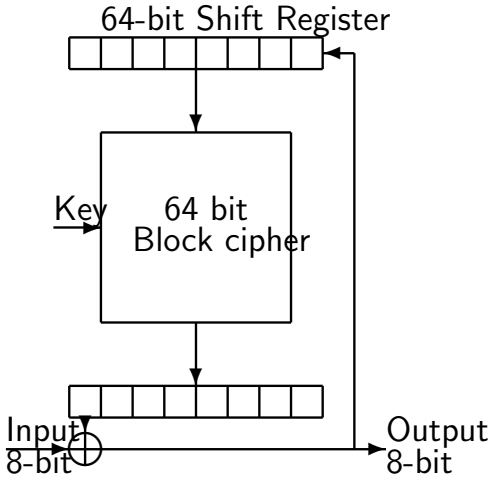
Thus, the attacker forces two cipherblocks to be identical.

Key Stream Generation Modes

In block encryption modes (like ECB and CBC), the point is to generate ciphertext that stores the message in encrypted but recoverable form.

In *key stream generation modes* the cipher is used more as a pseudorandom number generator. The result is a key stream that can be used for encryption by XORing with a message stream. Decryption uses the same key stream.

Cipher Feedback Mode



Cipher Feedback mode (CFB) allows for encryption of blocks smaller than block size. Each byte, say, is XORed with the first block of the previous output and fed back into the encryption.

Output Feedback Mode

Output Feedback Mode (OFB) is similar to CFB except that the quantity XORed with each plaintext block is generated independently of both plaintext and ciphertext, essentially by repeatedly encrypting the "seed."

Double DES

To address the weakness of the DES, some researchers have suggested applying the algorithm multiple times. Double encryption uses two keys and works as follows:

$$E(k_2, E(k_1, m)).$$

Naively, this should multiply the difficulty in breaking the algorithm. However, this is incorrect. Merkle and Hellman showed that two encryptions actually only double the time required to break DES at a cost of additional storage. This is equivalent to using a 57-bit key.

Aside on Double Encryption

Suppose you have any block cipher with a keysize of k. Suppose also that you have known plaintext/ciphertext pairs (P, C) and you are using the encryption

$$E(k_2, E(k_1, m))$$

where D is the corresponding decryption algorithm.

First compute and store in a hash table $E(k_i, P) = M_i$ for all 2^k keys k_i. Then successively compute the decryption $D(k_j, C) = M_j$ for each key k_j. As soon as you find $M_j = M_i$, you have a potential keypair $(k_2, k_1) = (i, j)$.

This has time complexity $O(2^k)$ but requires 2^k storage. This is called a *meet-in-the-middle* attack. It is a special case of a *time-memory tradeoff*.

Triple DES

Using *two* keys and *three* encryptions does strengthen the algorithm. The procedure is:

$$C = E(k_1, D(k_2, E(k_1, m))).$$

This effectively doubles the key length to a 112-bit equivalent key, at the cost of more complexity in key generation and management.

Of course, you can also use three independent keys, but this has the disadvantage of requiring more key generation and more key management.
Most variants are susceptible to attack using a time-memory tradeoff. But often the storage requirements are huge and the attack is infeasible.

Triple DES (cont.)

Using *three* keys and *three* encryptions further strengthens the algorithm. The procedure is:

$$C = E(k_1, D(k_2, E(k_3, m))).$$

This effectively yields an 168-bit equivalent key.

References to 3DES typically refer to this variant. It is believed that 3DES is strong. But it is slow. Why?

The request for proposals that led to AES explicitly mandated an algorithm that was "stronger and faster than 3DES."

Advanced Encryption Standard

In 1995, NIST began a search for a successor to DES. They called for an algorithm that was:

- unclassified;
- publicly disclosed;
- available royalty-free for use worldwide;
- symmetric block cipher algorithm for blocks of 128 bits;
- usable with key sizes of 128, 192, and 256 bits

Fifteen algorithms were selected for review. Five underwent extensive scrutiny and one was chosen as the new AES standard. The algorithm is the Rijndael algorithm of Dutch researchers Vincent Rijmen and Joan Daemen.

Overview of Rijndael

This fast algorithm uses substitution, transposition, and the shift, exclusive OR, and addition operators.

Like DES, AES uses repeat cycles or "rounds." There are 10, 12, or 14 cycles for keys of 128, 192, and 256 bits, respectively.

Overview of Rijndael

The text is arranged as a 4×4 array of bytes called the "state," which is modified in place in each round.

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

The key is also arranged as a $4 \times n$ array of bytes, and is initially expanded in a recursive process into $r + 1$ 128-bit keys, where r is the number of rounds.

Rounds in Rijndael

Each round consists of four steps.

subBytes: for each byte in the array, use its value as an index into a 256-element lookup table, and replace its value in the state by the byte value stored at that location in the table.

shiftRows: Let R_i denote the i^{th} row in state. Shift R_0 in the state left 0 bytes (i.e., no change); shift R_1 left 1 byte; shift R_2 left 2 bytes; shift R_3 left 3 bytes. This does not affect the individual byte values themselves.

Rounds in Rijndael

mixColumns: for each column of the state, replace the column by its value multiplied by a fixed 4×4 matrix of integers (as illustrated below).

$$\begin{bmatrix} a_0{}' \\ a_1{}' \\ a_2{}' \\ a_3{}' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

addRoundKey: XOR the state with a 128-bit round key derived from the original key K by a recursive process.

Decryption in Rijndael

The decryption algorithm is the inverse of encryption and has the following differences:

- The keys are used in reverse order.
- Each of the steps is inverted.
- The first and last rounds are slightly different.

Perhaps the biggest difference is the InverseMixColumns steps. The multiplication is by the fixed array:

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

For that reason, decryption typically takes longer than encryption.

Security of the AES

The algorithm is new and hasn't "stood the test of time." However, it was subjected to extensive analysis, and is incorporated in a large number of commercial encryption products.

Moreover, the developers were not related to the U.S. government, so there is little suspicion that the government weakened the algorithm or added a trapdoor.

No flaws have been discovered, but that doesn't mean that none exist.

Unlike DES, AES is modular and the key length can be extended if necessary. Similarly, the number of cycles can be increased.

Comparison of DES and AES

	DES	AES
Date	1976	1999
Block size	64 bits	128 bits
Key length	56 bits	128, 192, 256 (and possibly more) bits
Rounds	16	10, 12, 14 based on key length
Encryption primitives	substitution, permutation	substitution, shift, bit mixing
Cryptographic primitives	confusion, diffusion	confusion, diffusion
Design	open	open
Design rationale	closed	open
Selection process	secret	secret, but accepted public comment
Source	IBM, enhanced by NSA	independent Dutch cryptographers

Revisiting Key Length

Recall: For an n-bit block cipher with k-bit key, given a small number of plaintext/ciphertext pairs encrypted under key K, K can be recovered by exhaustive search in an expected time on the order of 2^{k-1} operations.

But, even in cases where 128-bit or larger keys are used with well-designed ciphers like AES, a brute force attack may be possible if keys are not generated properly (i.e., do not contain enough entropy).

Many commercial or shareware security products that advertise "128-bit security" derive keys from a user-selected password or passphrase. Users rarely employ passwords with anything close to 128-bit entropy. Such systems are often quite easy to break in practice.

Public Key Encryption

In a symmetric algorithm, the key must remain secret for the algorithm to be secure. In 1976, Diffie and Hellman proposed *public key encryption* (asymmetric encryption) in which a key does not have to be kept secret.

In 1997, it was publicly disclosed that asymmetric key algorithms were developed by James H. Ellis, Clifford Cocks, and Malcolm Williamson at GCHQ in the UK in the early 1970s. The researchers independently developed Diffie-Hellman key exchange and a special case of RSA. The GCHQ cryptographers referred to the technique as *non-secret encryption*.

Public Key Encryption

The idea is to use a publicly disclosed key to encrypt and a secret key to decrypt. This drastically reduces the number of keys have to be protected.

The requisite relationship is:

$$P = \{\{P\}_{K_{pub}}\}_{K_{priv}}.$$

Public Key Systems

Several alternative notations are in wide use for denoting the public and private keys of any principal. We'll use the following. The public key for principal A will be denoted by K_a and the private key will be denoted K_a^{-1}.

Thus, the relationship denoted previously as

$$P = \{\{P\}_{K_{pub}}\}_{K_{priv}}.$$

can also be written as:

$$P = \{\{P\}_{K_x}\}_{K_x^{-1}}$$

for any principal X. If the principal is assumed known, we may omit the subscript.

Public Key Systems

Also, for *some* public key systems, RSA in particular, E and D commute and either key can be used in either function. That is:

$$\{\{P\}_K\}_{K^{-1}} = P = \{\{P\}_{K^{-1}}\}_K.$$

This is crucial in some uses of RSA. *But is not true in general for public key cryptosystems.*

It's only "sort of" true for RSA (according to an expert I know) and assumes that the key doesn't have too many 1's in it's binary representation.

Public Key Systems

The basis of any public key system is the identification of a function that is easily computed, but difficult to invert without additional information. This is called a *one-way* function.

For example, it is straightforward to multiply two large primes p_1 and p_2 together. However, given the result p_1p_2, it is very difficult to factor it to recover the p_1 and p_2.

However, given p_1p_2 and either of p_1 or p_2, it is again straightforward to recover the other, simply by dividing.

Thus, multiplication is a one-way function, because computing the inverse is effectively infeasible for products of large primes.

Authentication with Public Keys

If A receives a message encrypted with his public key, only he can decrypt it. However, even if the message claims to come from B, there is no assurance that it does so since C also has access to A's public key.

Authentication can be gained with *some* public key system (as with RSA where encryption and decryption commute). If A receives a message purportedly from B, encrypted with B's *private* key, and A can decrypt it with B's public key, then the message almost certainly came from B.

We can think of encryption as a *privacy* transformation, and decryption as an *authenticity* transformation.

RSA Algorithm

The **Rivest-Shamir-Adelman (RSA)** algorithm is a public key system which relies on the difficulty of factoring large numbers.

Two keys, d and e, are used for encryption and decryption. Because the use of the keys in RSA is symmetric, either can be the private key. The algorithm is such that:

$$\{\{P\}_d\}_e = P = \{\{P\}_e\}_d.$$

A plaintext block P is encrypted as (P^e mod n). The decrypting key is chosen so that: $(P^e)^d \bmod n = P$.
Because the encrypting exponentiation is performed modulo n, an interceptor would have to factor P^e to recover the plaintext. This is difficult. However, the legitimate receiver knows d and merely computes $(P^e)^d \bmod n = P$. This is much easier.

Other Public Key Algorithms

A public key system can be based on any one-way function. A rich source of these is the set of NP-complete problems. These are infeasible to solve (requiring at least exponential time) but a solution can be checked in polynomial time.

Merkle and Hellman proposed a public key system based on the *knapsack problem*: given a set of integers and a target sum, find a subset of the integers that sum to the target.

The algorithm is theoretically very secure, but has practical weaknesses. Just because a problem is difficult, in general, doesn't mean it is difficult in specific instances.

Efficiency of Encryption

Public key systems simplify key distribution but are typically very slow. Thus, they are used only for specialized purposes.

A public key encryption *may take 10,000 times as long* to perform as a symmetric encryption, because it depends on multiplication and division, rather than simple bit operations.

For this reason, symmetric encryption is the work horse of commercial cryptography. Still, asymmetric encryption plays some important functions.

Key Size Revisited

From *Introduction to Cryptography*, by PGP Corporation:

> *...public key size and conventional cryptography's secret key size are totally unrelated. A conventional 80-bit key has the equivalent strength of a 1024-bit public key. A conventional 128-bit key is equivalent to a 3000-bit public key. Again, the bigger the key, the more secure, but the algorithms used for each type of cryptography are very different and thus comparison is like that of apples to oranges.*

Applications of Cryptography

Now we consider some applications of cryptography.

- **Cryptographic Hash Functions**
- Key Exchange
- Digital Signatures
- Certificates

Hash Functions

Given a file, we compute a cryptographic function, called a **hash** or **checksum** or **message digest**.
Ideally, the function should have these qualities:

- it is easy to compute the hash for any given data,
- it is extremely difficult to construct a text that has a given hash,
- it is extremely difficult to modify a given text without changing its hash,
- it is extremely unlikely that two different messages will have the same hash.

Vocabulary

A function f is *preimage resistant* if, given h, it is hard to find any m such that $h = f(m)$.

A function f is *second preimage resistant* if, given an input m_1, it should be hard to find $m_2 \neq m_1$ such that $f(m_1) = f(m_2)$. This is sometimes called *weak collision resistance*.

A function f is (strong) *collision resistant* if it is hard to find two messages m_1 and m_2 such that $f(m_1) = f(m_2)$.

Birthday Attacks

If a function $f(x)$ yields any of H different outputs with equal probability and H is sufficiently large, then we expect to obtain a pair of different arguments x_1 and x_2 with $f(x_1) = f(x_2)$ after evaluating the function for about $1.25\sqrt{H}$ different arguments on average.

What does this mean for a hash value of 128 bits? for 160 bits?

Cryptographic Hash Functions

We usually use encryption to conceal the contents of an object, i.e., to protect confidentiality. However, in some cases integrity is the desired result.

- In a document retrieval system containing legal records, it may be important to know that the copy retrieved is identical to that stored.
- In a secure communications system, the correct transmission of messages may override confidentiality concerns.

A cryptographic hash function "binds" the bytes of a file together in a way that makes any alterations to the file apparent. Cryptography is used to *seal* the file to make it tamper-proof or, at least, tamper-resistant.

Hash Functions (Cont.)

The process is as follows: given a sensitive file f, compute the hash function and store it with the file. Each time the file is used or accessed, recompute the hash. If the stored value matches the newly computed value, it is likely that no changes have occurred.

An encryption algorithm such as DES or AES can be used to generate a hash value. The encryption itself yields a checksum since it seals the file against modification. However, it fails the criterion of being small.

Hash Functions (Cont.)

Instead, suppose you use DES to encrypt successive blocks of plaintext. As each block's ciphertext is computed, XOR it with the running hash of the previous blocks. The file's checksum is then the final result of the chained encryption.
This scheme has the following properties:

- The checksum depends on all blocks in the file.
- It is virtually impossible to modify the file in such a way as to preserve the checksum (i.e., it is *non-malleable*).

Common Hash Algorithms

The most widely used cryptographic hash functions are:

- MD4: (Message Digest 4) invented by Ron Rivest and RSA Laboratories;
- MD5: an improved version of MD4;
- SHA/SHS: (Secure Hash Algorithm or Standard) similar to MD4 and MD5.

MD4 and MD5 both compress a message of any size to a 128-bit digest. SHA/SHS produces a 160-bit digest.

Applications of Cryptography

- Cryptographic Hash Functions
- **Key Exchange**
- Digital Signatures
- Certificates

Key Exchange

Suppose you want to establish a secure communication channel with someone you do not know and who does not know you. We call this a situation of *mutual suspicion*. This is extremely common.

- You submit your income tax on-line.
- You send your credit card information to a shopping website.
- You wish to exchange encrypted email with another party.

Once you agree on a shared secret (key) the communication can proceed. But how do you exchange the key? This is an issue of *cryptographic protocol design.*

Key Exchange (Cont.)

Suppose both parties S and R have a public / private RSA key pair for asymmetric communication. One protocol is for one party, say S, to choose a new symmetric key K and send it to R in the form:

$$\{K\}_{K_S^{-1}}.$$

R can decrypt the message using S's public key to retrieve K.
What is wrong with this scheme?

Key Exchange (Cont.)

Suppose both parties S and R have a public / private RSA key pair for asymmetric communication. One protocol is for one party, say S, to choose a new symmetric key K and send it to R in the form:

$$\{K\}_{K_S^{-1}}.$$

R can decrypt the message using S's public key to retrieve K.
What is wrong with this scheme?

Answer: Any eavesdropper can intercept the message and decrypt it using S's public key to retrieve K.

Key Exchange (Cont.)

As a second attempt, suppose S sends

$$\{K\}_{K_R}$$

to R. Since only R can decrypt this message, confidentiality is assured. However, something is still amiss.

Key Exchange (Cont.)

As a second attempt, suppose S sends

$$\{K\}_{K_R}$$

to R. Since only R can decrypt this message, confidentiality is assured. However, something is still amiss.

Now R doesn't know have any assurance that the message actually came from S. An intruder may be "spoofing" (pretending to be S) to obtain information that R intends only for S.

Can we preserve both confidentiality and authentication with one transaction?

Key Exchange (Cont.)

A third attempt is for S to send R the following:

$$\{\{K\}_{K_S^{-1}}\}_{K_R}.$$

R decrypts the message using K_R^{-1} to reveal the contents, $\{K\}_{K_S^{-1}}$.
What assurances does this provide?

Key Exchange (Cont.)

A third attempt is for S to send R the following:

$$\{\{K\}_{K_S^{-1}}\}_{K_R}.$$

R decrypts the message using K_R^{-1} to reveal the contents, $\{K\}_{K_S^{-1}}$. What assurances does this provide?

Since, no one but R can decrypt the message, confidentiality is assured.

R then decrypts the inner message using K_S to obtain K. No one but S could have performed the inner encryption, so authentication is accomplished.

This notion of nested encryptions is very useful in a variety of cryptographic protocols. Could you have done the encryptions in the other order?

Diffie-Hellman Key Exchange

The question of key exchange was one of the first problems addressed by a cryptographic protocol.

The Diffie-Hellman key agreement protocol, invented in 1976, was the first practical method for establishing a shared secret over an unsecured communication channel. (It later emerged that it had been invented somewhat earlier by GCHQ, the British intelligence agency.)

The point is to agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key.

Diffie-Hellman Key Exchange (cont.)

Steps in the algorithm:

1. Alice and Bob agree on a prime number p and a base g.
2. Alice chooses a secret number a, and sends Bob $(g^a \mod p)$.
3. Bob chooses a secret number b, and sends Alice $(g^b \mod p)$.
4. Alice computes $((g^b \mod p)^a \mod p)$.
5. Bob computes $((g^a \mod p)^b \mod p)$.

Both Alice and Bob can use this number as their key.

Diffie-Hellman Example

1. Alice and Bob agree on $p = 23$ and $g = 5$.
2. Alice chooses $a = 6$ and sends $5^6 \mod 23 = 8$.
3. Bob chooses $b = 15$ and sends $5^{15} \mod 23 = 19$.
4. Alice computes $19^6 \mod 23 = 2$.
5. Bob computes $8^{15} \mod 23 = 2$.

Then 2 is the shared secret.

A nice video describing this is here: `http://www.wimp.com/howencryption/`.

Diffie-Hellman Security

Clearly, much larger values of a, b, and p are required. An eavesdropper cannot discover this value even if she knows p and g and can obtain each of the messages.

Suppose p is a prime of around 300 digits, and a and b at least 100 digits each. Finding a given g, p, and $g^a \mod p$ would take longer than the lifetime of the universe, using the best known algorithm. This is called the *discrete logarithm problem*.

Applications of Cryptography

- Cryptographic Hash Functions
- Key Exchange
- **Digital Signatures**
- Certificates

Digital Signatures

Consider the process of transferring funds from one person to another. Conventionally, you write a check.

- A check is a *tangible object* authorizing the transaction.
- The signature on the check *confirms authenticity*.
- In the case of an alleged forgery, a third party may be called to *judge authenticity*.
- The check is *not alterable* or, at least, alterations are easily detected.

Transactions on computers do not generate tangible objects. But can we define a mechanism for signing a document digitally that has analogous characteristics?

Digital Signatures (Cont.)

We'd like a digital signature to have analogous properties:

unforgeable: If P signs message M with signature $S(P, M)$, it must be impossible for anyone else to produce $S(P, M)$. Also ensures *no repudiation.*

authentic: If R receives pair $[M, S(P, M)]$, purportedly from P, R can check that the signature really is from P. Only P could have created this signature.

tamperproof: After being transmitted, M cannot be changed by S, R, or an interceptor.

not reusable: The signature cannot be detached from the message and reused for another message.

Digital Signatures (Cont.)

Public key systems are well-suited for digital signatures. Recall that some algorithms, RSA in particular, have the following characteristic:

$$\{\{M\}_K\}_{K^{-1}} = M = \{\{M\}_{K^{-1}}\}_K.$$

So, if S wishes to send message M to R in a way that has some of the characteristics of a digitally signed message, S could send

$$\{\{M\}_{K_S^{-1}}\}_{K_R}.$$

Most often, it's not the M but a hash of M that is signed. Why?

What assurance does R gain from this interchange? In particular, which of: unforgeable, authentic, tamperproof, non-reusable?

Digital Signatures (Cont.)

This scheme gives all of the desirable characteristics.

1. *Authenticity / Unforgeability*: because the message can be decrypted only with S's public key, it must have come from S.
2. *Non-repudiation*: R saves $\{\{M\}_{K_S^{-1}}\}_{K_R}$. If S later tries to disavow sending the message, anyone can verify that only this is transformed to M with S's public key. Thus, only S could have sent it.
3. *Tamperproof*: because the only R can remove the outer layer of encryption. What if you sign a hash instead of the message?
4. *Non-reusable*: because the signature is not a separate piece of data, but is intimately bound to the message. Is that still true if you sign a hash?

Applications of Cryptography

- Cryptographic Hash Functions
- Key Exchange
- Digital Signatures
- **Certificates**

Certificates

Certification addresses the need for *trust* in computer systems. How do two entities that are mutually suspicious establish a relationship of trust. One way is to rely on a third party who "vouches for" the trustworthiness of one or both of the parties.

We may believe a party's affiliation or ask for independent validation. In general, we have a "trust threshold," a degree of trust we're willing to confer without going further in the certification process. This threshold may depend on the size or nature of the transaction.

The police, Chamber of Commerce, Better Business Bureau, and credit reporting agencies all function in part as certification authorities for such transactions.

Need for Trust

The most common circumstance in which trust is needed in a distributed context is in *binding a key to an identity*.

That is, how do I know that the public key you present is really your public key and not someone else's?

Certificates (Cont.)

Establishing trust may involves "chains" of certification.

In a large company, your supervisor may vouch for or certify your employment. His supervisor may vouch for him, and so on. A truly paranoid customer may require a chain of certifications leading back to some unimpeachable authority at the base of the chain, such as the president of the company, before dealing directly with you.

However, it would be unmanageable to require all of these parties to be present for communication to occur. There is a need securely to store and pass around records of such certification.

Certification (Cont.)

Sometimes certification occurs through a *common respected individual*. For example, suppose Ann and Andrew work for different divisions within the same company. Presumably, they have a common supervisor (ancestor in the hierarchy tree of the company).

If both trust their management, they can certify each other's authenticity via their common supervisor.

The chain can begin at the top or from the bottom of the hierarchy.

Certificates (Cont.)

Electronically, certification is accomplished with digital signatures and hash functions.

A public key and user's identity are bound together in a **certificate**. This is then signed by a **certification authority**, vouching for the accuracy of the binding.

Certificates (Cont.)

The following are possible steps. Suppose X is the president of the company and her immediate subordinate is Y. Each have a public key pair.

1. X publishes K_X to all employees.
2. Y securely passes message $M = \{Y, K_Y\}$ to X.
3. X produces a hash H of the message, i.e., $h(\{Y, K_Y\})$.
4. X produces $\{M, \{H\}_{K_X^{-1}}\}$.

This last then becomes Y's **certificate**.

What has been accomplished?

Certificates (Cont.)

Y's certificate is X's affirmation of her identity. Anyone can decrypt it with X's public key and look at the contents. Only X could have produced it. The hash ensures that it was not altered.

Now Y can certify the identify of her subordinates in a similar manner. She appends her certificate to each of theirs. This provides a chain of validations back to X.

Thus, an individual's certificate contains a chain of evidence rooted at some unimpeachable authority.

Certificates (Cont.)

There is also a need for trust in situations where there is not a single hierarchy, such as on the Internet. Two individuals may not have a common "superior." Some entity may be designated as a certification authority (notary public, personnel office, security officer in a company, etc.).

On the Internet, several groups serve as "root certification authorities": C & W HTK, SecureNet, Verisign, Baltimore Technologies, Deutsche Telecom, Certiposte, and several others. These tend to be structured around national boundaries.

X.509 Certificates

The International Telecommunications Union (ITU) has issued a standard (X.509) for digital certificates that is the basis for many other protocols. An X.509v3 certificate has the following components:

1. *Version:* version of X.509 used;
2. *Serial number:* unique among certificates issued by this issuer;
3. *Signature algorithm identifier:* identifies the algorithm and params used to sign the certificate;
4. *Issuer's distinguished name:* with serial number, makes all certificates unique;
5. *Validity interval:* start and end times for validity;
6. *Subject's distinguished name:* identifies receiver of the certificate;
7. *Subject's public key info:* identifies algorithm, params, and public key;

X.509 Certificates (Cont.)

8. *Issuer's unique id:* used if an Issuer's distinguished name is ever reused;
9. *Subject's unique id:* same as field 8, but for the subject;
10. *Extensions:* version specific information;
11. *Signature:* identifies the algorithm and params, and the signature (encrypted hash of fields 1 to 10).

To validate the certificate, the user obtains the issuer's public key for the algorithm (field 3) and deciphers the signature (field 11). Then uses the information in the signature field (field 11) to recompute the hash and compare with the received value. Finally, check the validity interval.