

Introduction To Cryptography

Traditional use of cryptography:



cryptographer: invent clever secret codes.

cryptanalyst: attempt to break these codes.

Fundamental Tenet of Cryptography:

If lots of smart people failed to solve a problem, then it probably won't be solved (soon).

- The time required to break a code should be longer than the time the encrypted data must remain secret.
- The value of most data *decreases* overtime.

Cryptographic System: *Algorithm* + *Key*

It is perfectly OK to let everyone know the algorithm because knowledge of the algorithm without the key does not help unmanle the information.

Publishing the algorithm provides an enormous amount of free consulting to uncover weaknesses.

Computational Difficulty:

Example: *combination lock*

- Typically require 3 numbers between 1 and 40.

If it takes 10 seconds for a good guy,
it would take $10 \cdot (40^{**3})$ seconds or about 1 week for the bad
guy.

➤ By requiring 4 numbers,
If it takes 13 seconds for the good guy,
it would take $13 \cdot (40^{**4})$ seconds or about 1 year for the bad
guy!

In general, increasing the key length by 1 bit makes the good guy's
job just a little bit harder, but makes the bad guy's job twice as
hard!

Example of Secret Codes:

Caesar cipher: substitute each letter with another letter which
is 3 letters away in the alphabet (with wrap around).

E.g., dozen >>> grchq.

Extension: Instead of 3 use any number n between 1 and 25.

E.g., for n=1, HAL >>> IBM.

More Examples

Monoalphabetic cipher: arbitrary map one letter to another.
There are $26! = 4 \cdot (10^{**26})$ possibilities.

- If each possibility takes 1 microsecond it would take 10
trillion years to try all possibilities.
- However *statistical analysis* of language makes it much
easier to break:

Examples

The Vigenère cipher: A different row of the *alphabet square* can
be used to encrypt each letter of the message. A key is used to
define the rows.

Examples

The Homophonic Substitution cipher: Replacing each letter with a *variety of substitutes*, the number of potential substitutes being proportional to the *frequency of the letter*.

Examples

Breaking An Encryption Scheme

Ciphertext Only: Because it is important for Trudy to be able to differentiate plaintext from gibberish, this attack sometime known as *recognizable plaintext attack*.

It is also essential for Trudy to have **enough ciphertext**, e.g., there is no way to know the plaintext corresponding to **XYZ** is **THE** or **CAT** or **HAT**.

Known Plaintext: Knowing *<plaintext, ciphertext>* pairs is helpful for Trudy, e.g., in monoalphabetic cipher Trudy learn the mapping.

How to obtain plaintext? secret data might not remain secret forever, e.g., which city to attack might not be secret after that attack!

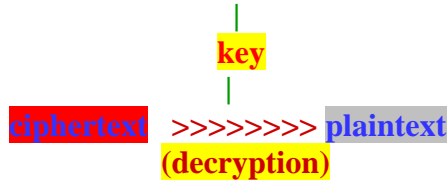
Chosen Plaintext: Trudy can choose any plaintext he wants and get the system to provide corresponding ciphertext, e.g., telegraph company can encrypt and send any message you provide.

Example: If Trudy knows that Alice encrypted message (**azsxopfq**) is either (**surrender**) or (**fight on**), he can ask the system to encrypt both messages and find out the correct one.

Types of Cryptographic Functions:

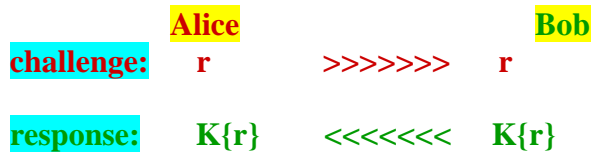
- **Secret Key Cryptography (symmetric cryptography)**

plaintext >>>>>>>>> (encryption) ciphertext



Can be used for:

- **Transmission Over an Insecure Channel:**
An eavesdropper will only see unintelligible data.
- **Secure Storage on Insecure Media:**
Forgetting the key makes the data irrevocably lost.
- **Authentication:** Alice authenticating Bob:



r is a random number,

K{r} is the secret key encryption of **r** using shared key **K**.

Public Key Cryptography (asymmetric cryptography)

Each individual has **two keys**:

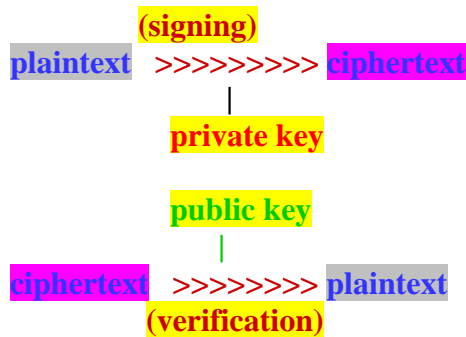
private key (not revealed to anyone)

public key (make it known to everyone)





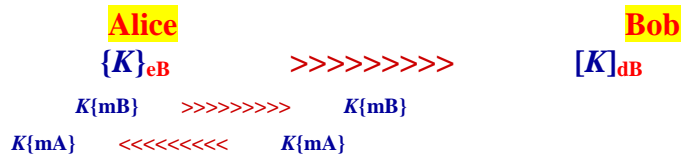
The reverse process is called digital signature:



- Public key cryptographic algorithms are *orders of magnitude slower* than the best known secret key cryptographic algorithms.
- Thus they normally used to establish *temporary shared secret* key for use during a session.

Uses of Public Key Cryptography:

- **Transmission Over an Insecure Channel:**



K is a random session key.

eB, dB is the *<public-key , private-key>* pair for Bob.

$\{m\}_{eB}$ is the public key *encryption* of message m with eB .

$[m]_{dB}$ is the public key *decryption* of message m with dB .

$K\{m\}$ is the *symmetric* key encryption of message m with K .

mB message for Bob.

mA message for Alice.

○ **Secure Storage on Insecure Media:**

Alice generates a random key K and save:

$$F = K\{\text{File}\}$$

$$KF = \{K\}_{eA}$$

To restore the file:

$$K = [KF]_{dA}$$

$$\text{File} = K\{F\}$$

○ **Authentication: Alice authenticating Bob:**

	Alice		Bob
challenge:	$c = \{r\}_{eB}$	>>>>>	c
response:	r	<<<<<	$r = [c]_{dB}$

r is a random number

• **Hash Algorithms**

(also known as message digest /fingerprint, one-way functions)

The hash of a message m , $h=H(m)$ has the following properties:

- Given m , it is **easy** to compute h .
- Given h , it is **hard** to compute m .
- Given m , it is **hard** to find another m' such that $H(m) = H(m')$.
- It is **hard** to find $m1$ and $m2$ such that $H(m1) = H(m2)$.

Uses of Hash Algorithms:

○ MAC/MIC (Message Authentication/Integrity Code)

Using Secret Key:

Alice sends

Bob receives

m, h where $h = H(m|K)$ >>>>> m, h , OK if $h = H(m|K)$

K is the shared secret between Alice and Bob

- Bob is sure that Alice sent the message, since she knows **K**.
- Bob *can NOT prove* to any one that Alice sent him message **m**, since he also knows **K**.

Using Public Key:

Alice sends

Bob receives

m, sh where $sh = [H(m)]_{dA}$ >> m, sh , OK if $H(m) = \{sh\}_{eA}$

$\langle eA, dA \rangle$ is the public/private key pair of Alice

- Bob is sure that Alice sent the message, since she knows **dA**.
- He **can also prove** to any one that Alice sent him message **m**, since he does not know **dA**.

This property of public key cryptography is known as *non-repudiation*,

where the sender should not be able to falsely **deny** that he sent a message.

○ Password Hashing:

OS like UNIX stores the *hash of passwords* instead of storing the *actual passwords*.

- For each user **U**, there is a tuple $\langle U, h \rangle$ where $h = H(P)$ is the hash of password **P** of user **U**.
- When a user **U** types a password, **P**, the OS compute $H(P)$ and if it is equal to the saved value h in the tuple $\langle U, h \rangle$, the user is OK.

The magic of XOR:
A Simple XOR symmetric algorithm:

(from [Bruce Schneier](#) textbook)

$$\begin{aligned} 0 \otimes 0 &= 0 \\ 0 \otimes 1 &= 1 \\ 1 \otimes 0 &= 1 \\ 1 \otimes 1 &= 0 \end{aligned}$$

Note that:

$$\begin{aligned} x \otimes x &= 0 \\ x \otimes 0 &= x \end{aligned}$$

The following program is a very simple symmetric algorithm.
 (see /home/cs772/public_html/demos/xor)

- To encrypt: the **plaintext** P is XORed with a key K to produce a **ciphertext** C .
- To decrypt: the **ciphertext** C is XORed with a key K to produce a **plaintext** P .

$$\begin{aligned} P \otimes K &= C \\ C \otimes K &= P \end{aligned}$$

Why?

$$\text{Since } C \otimes K = (P \otimes K) \otimes K = P \otimes (K \otimes K) = P \otimes 0 = P$$

This is a very insecure algorithm that can be easily broken.
 It is good enough to keep "your kid sister" from reading your files!

crypto

Usage:

```
% crypto <key> <input_file> <output_file>
```

Example:

- To encrypt infile:

```
% crypto "odu computer science" infile outfile
```

- To decrypt outfile:

```
% crypto "odu computer science" outfile tmpfile
```

tmpfile is identical to infile

Code

```
void main (int argc, char *argv[])
{
    FILE *fi, *fo;
    char *cp;
    int c;

    if ((cp = argv[1]) && *cp!='\0') {
        if ((fi = fopen(argv[2], "rb")) != NULL) {
            if ((fo = fopen(argv[3], "wb")) != NULL) {
                while ((c = getc(fi)) != EOF) {
                    if (!*cp) cp = argv[1];
                    c ^= *(cp++);
                    putc(c,fo);
                }
                fclose(fo);
            }
            fclose(fi);
        }
    }
}
```