

Cryptography

Cryptography

Cryptography is a rich field

- with a long history,
- more recently, with interesting mathematics,
- with several branches and specialties
(steganography, cryptanalysis, algorithms, ...)
- with its own conferences, journals, buzzwords, ... ,
- with important applications in security.

We will cover only some topics in cryptography, at a fairly high-level.

We will do a bit more later in the course.

There is more that you should (and may already) know. Please see the Handbook of Applied Cryptography and other references.

Bits of wisdom (?)

Cryptography is not broken, it is circumvented.
(Shamir according to Rivest)

Bits of wisdom (cont.) (?)

If you think that cryptography is the answer to your problem then you don't understand cryptography and you don't understand your problem.

(Needham or Lampson)

Bits of wisdom (cont.) (?)

Cryptography is nothing more than a mathematical framework for discussing the implications of various paranoid delusions.

(Alvarez according to Guttman)

Cryptography and security

Cryptography is not the same as security.

But the security applications of cryptography are so significant that they have shaped both fields.

The definitions of correctness for cryptographic constructions are informed by security applications.

Modern cryptography typically considers not just algorithms but also models of protocols and systems.

Cryptography and encryption

There is much more to cryptography than encryption.

But looking first at encryption is fair and instructive, so we will.

We will start with **shared-key encryption**, also called **symmetric encryption**.

The same keys are used for encrypting and decrypting.

Shared-key encryption



K is a key.

Both E and D have the key.

E and D may or may not be done by the same principal.

K should be secret.

E and D may not be secret.

Shared-key encryption (cont.)

The main goal of encryption is secrecy.

When f is a shared-key encryption function, a ciphertext $f(K, M)$ should reveal nothing—or as little as possible—about the plaintext M to someone who does not have K .

(A more precise definition will come later.)

Shared-key encryption methods

Substitution ciphers

easy to understand

easy to run

easy to deploy

easy to break

Shared-key encryption methods (cont.)

XORing with one-time pads

easy to understand

just a little harder to run

hard to deploy

hard to break (in theory)



Each key K can be used for only one piece of data.

Shared-key encryption methods (cont.)

Modern methods

harder to understand

moderately hard to run

easier to deploy

hard to break (we believe)

(To be continued.)

Some concerns (summary)

Key distribution

Execution complexity

Security

Types of attacks

Ciphertext only

Known plaintext

Chosen plaintext (with ciphertext)

Chosen ciphertext (with plaintext)

Types of attacks (cont.)

Obtaining key material somehow

- looking through backup tapes and dumpsters
- through a software virus
- by breaking the key generator
- by social engineering
- by “rubber-hose cryptanalysis”

(threats, blackmail, torture, and bribery)

Types of attacks (cont.)

Side-channel analysis

- power
- timing
- electromagnetic radiation
- ;

Power analysis (from Kocher, Jaffe, and Jun)

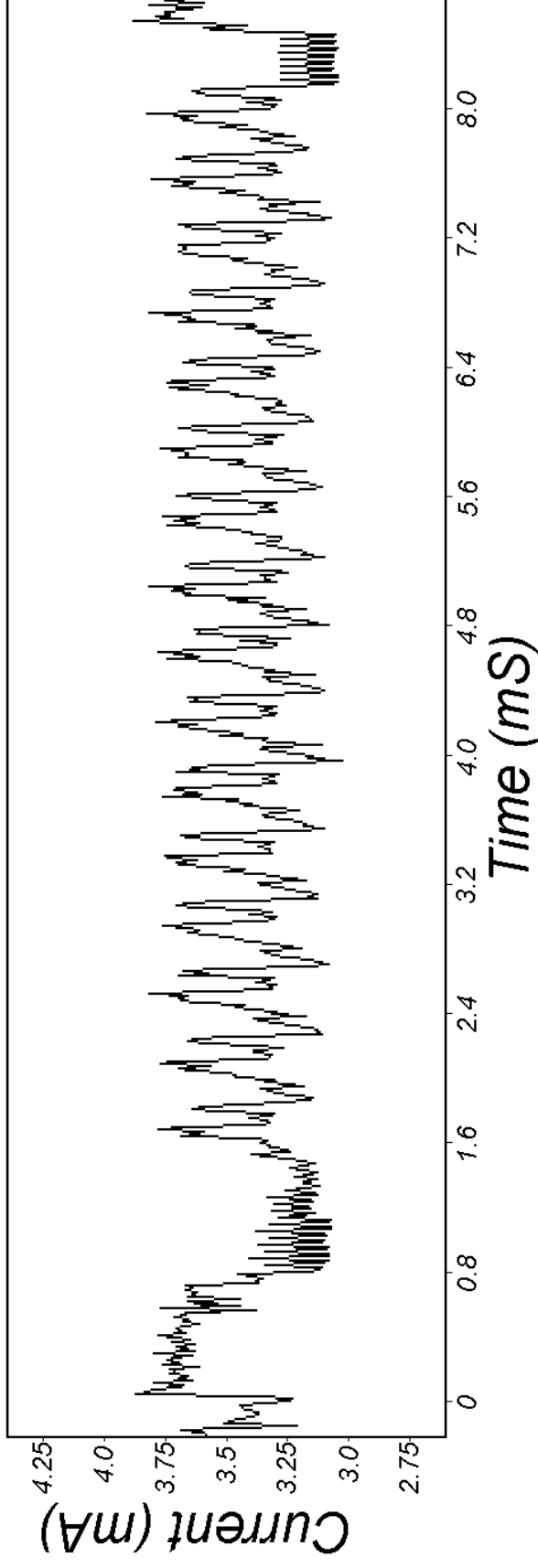


Figure 1: SPA trace showing an entire DES operation.

A *trace* refers to a set of power consumption measurements taken across a cryptographic operation. For example, a 1 millisecond operation sampled at 5 MHz yields a trace containing 5000 points. Figure 1 shows an SPA trace from a typical smart card as it performs a DES operation. Note that the 16 DES rounds are clearly visible.

Themes

Arms race between cryptographers and cryptanalysts

- with occasional disconnects (e.g., Mary Queen of Scots uses monoalphabetic ciphers long after they are known breakable),
- often active during wars.

Multi-disciplinary field:

- linguistics,
- mathematics,
- computing,
- physics.

Suspicion about new algorithms.

Increased confidence from cryptanalysis efforts.

Approximating the one-time pad

- Start with a fixed-size key.
- Stretch it to obtain a key stream as long as the plaintext, with shifts, XORs, etc..
- The key stream should “look random” .
- XOR the key stream with the plaintext.

This is a **stream cipher**.

As in this case, shared-key cryptosystems are typically based on fast, low-level operations.

Another approach

Block ciphers apply keys of fixed length to plaintext blocks of fixed length. They are iterated for encrypting to arbitrary plaintexts.

The most famous is probably DES.

It has shown cracks, particularly in the last few years.

Its successor is AES.

(You should know about DES and AES, from the reading if necessary.)

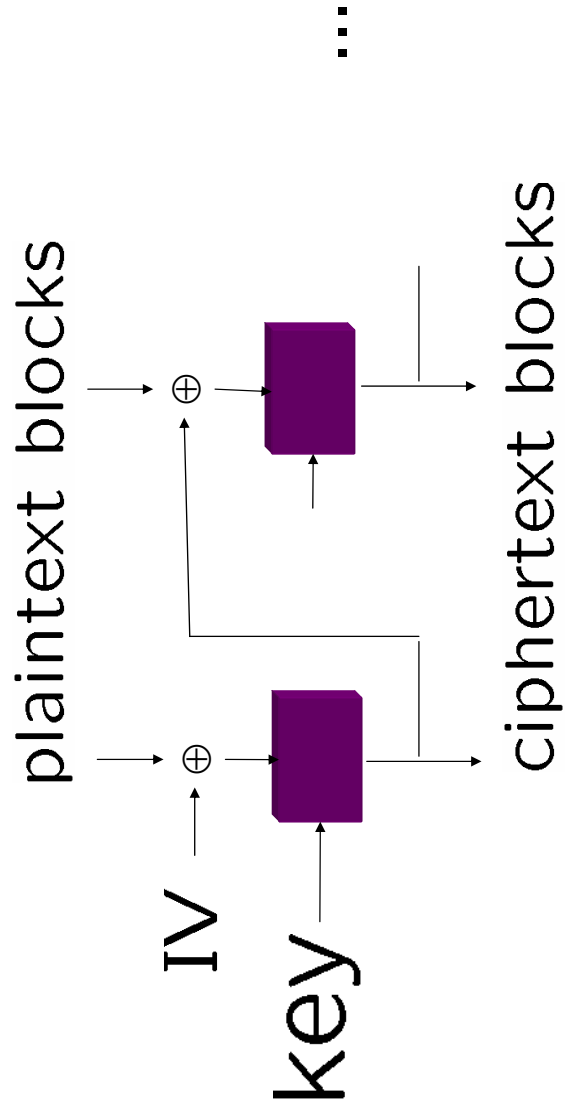
Two of the modes of operation

ECB (electronic code book): long plaintexts are encrypted block by block, each block independently.

- Blocks can be exchanged (no integrity).
- Block equalities leak (no secrecy).

CBC (cipher block chaining): encryptions are chained

CBC



Multiple encryption

Now suppose that we want to increase the key length.

We may try to let the ciphertext be obtained by encrypting the plaintext under several keys in sequence.

Or we may use an encryption algorithm that takes longer keys, by definition (simpler and better when feasible).

An attack on double encryption

ciphertext = encryption of plaintext under K_1 then K_2



There should be only one or few key pairs that yield a match (for large enough block size).
All but one of them can be discarded by checking a few other pairs plaintext/ciphertext.

Cost: two iterations over the key space and a large table

Probabilistic encryption

Encryption can be randomized.

That is, E may take a third argument.

Thus, two encryptions of a plaintext with a key need not be identical.

Public-key encryption

Public-key encryption generalizes shared-key encryption:

- Each principal has a secret key for decrypting.
- The inverse of the secret key is a public key for encrypting.

$$D(K, E(K^{-1}, M)) = M, \text{ for a legal pair } (K, K^{-1}) \text{ and } M.$$

Public-key encryption is called **asymmetric encryption**.

It usually relies on more mathematics, can be slow.

Public-key encryption: features

With shared-key encryption, we face a big key distribution problem:

- There is a key for each pair of principals.
- If not preestablished, the keys are known to the key distribution center.

So the key distribution center needs to be very trusted.

In the public-key world, key distribution should go better.

- There is a key pair for each principal.
- The public key can be published.
- If not preestablished, the key distribution center needs only the public key.

(Much more on key distribution later!)

Public-key encryption: RSA

Encryption key:

- a modulus $N = pq$, where p and q are two (randomly chosen) primes,
- an exponent e that has no factors in common with $p - 1$ or $q - 1$.

Decryption key: the factors p and q .

$$E((N, e), M) = M^e \pmod{N}$$

$$D((p, q), C) = C^d \pmod{N} \text{ where } ed \equiv 1 \pmod{(p-1)(q-1)}$$

Public-key encryption: RSA (cont.)

With a little number theory:

- d can be found efficiently:
given e and $(p-1)(q-1)$, the gcd algorithm can find d and k such that $ed + k(p-1)(q-1) = 1$;
- $C^d \equiv M^{ed} \equiv M^{1-k(p-1)(q-1)} \equiv M \pmod{N}$.

RSA vs. factoring

Breaking RSA seems to be very hard.

Factoring seems to be very hard.

Breaking RSA is no harder than factoring (but it may be easier).

A weakness of pure RSA

Given

$$E((N, e), M_1) = M_1^e \bmod N$$

$$E((N, e), M_2) = M_2^e \bmod N$$

an attacker can compute

$$\begin{aligned} E((N, e), M_1 \times M_2) &= (M_1 \times M_2)^e \bmod N \\ &= E((N, e), M_1) \times E((N, e), M_2) \bmod N. \end{aligned}$$

This homomorphism is destroyed in standards based on RSA (e.g., OAEP).

Hybrid encryption

Encrypting with a system like RSA can be slow.

For speed, one can use a hybrid scheme,
for example:

- encrypt a fresh AES key under an RSA key,

- encrypt the plaintext under the AES key.

Diffie-Hellman

Let p be a prime and g a generator of Z_p^* (chosen with a little care).

A invents x and publishes $g^x \bmod p$.

B invents y and publishes $g^y \bmod p$.

x and y serve as secret keys.

$g^x \bmod p$ and $g^y \bmod p$ as public keys.

Both A and B can compute $g^{xy} \bmod p$.

It is a shared secret—with someone!

(It is not authenticated.)

From the shared secret, A and B can for example compute keys.

Diffie-Hellman vs. discrete logarithms

Computing discrete logarithms is hard—so presumably others can't learn the shared secret.

But there is no proof that breaking Diffie-Hellman is as hard as computing discrete logarithms.

Other cryptographic operations

One-way functions (e.g., encryption).

$f(M)$ is easy to compute but hard to invert:
given $f(M)$, it is hard to find N such that $f(M) = f(N)$.

One-way hash functions (e.g., SHA).

In addition, $f(M)$ is short, but it is hard to find M and N with $M \neq N$ and $f(M) = f(N)$.

Practitioners often assume that $f(M)$ looks random.

Lately some flavors of SHA have been attacked with some success.

One-way hash functions are often defined by iterating a basic compression function, e.g.,

$$f(M_1M_2) = h(M_1, M_2)$$

$$f(M_1 \dots M_{n+1}) = h(f(M_1 \dots M_n), M_{n+1}) \text{ for } n > 1$$

where all M_i 's are of some fixed length.

They can be quite fast.

What about integrity?

MACs (e.g., HMAC-SHA):

- Two principals know a key.
- Both principals use the key for signing.

A MAC is a **message authentication code**.

A MAC $f(K, M)$ should be easy to compute from K and M , but hard to compute without knowing K .

Properties of good MACs

Ease of computation: Computing $f(K, M)$ is fast.

Compression: $f(K, M)$ is short—ideally of a small constant size.

Computation-resistance: Given $f(K, M_1), \dots, f(K, M_n)$, it is hard to compute $f(K, M)$, for a new M .

\Rightarrow So $f(K, M_i)$ should not leak K , but it may reveal M_i .

Constructing MACs

Typically, MACs are based on hash functions and on encryption functions.

For example, we may try to define a MAC function f from a one-way hash function g by:

$$f(K, M) = g(K M)$$

But this is subject to an **extension attack**, since:

$$f(K, M_1 \dots M_{n+1}) = h(f(K, M_1 \dots M_n), M_{n+1})$$

if g is defined from the compression function h .

There are better ideas, for example:

$$f(K, M) = g(K g(K M))$$

Public-key signatures (e.g., RSA):

- Each principal has a secret key for signing.
- The inverse of the secret key is a public key for checking signatures.

For speed, one typically signs a hash of the plaintext rather than the full plaintext.

The random

Generating good (pseudo)random numbers is crucial for cryptography.

- With them, we have at least the one-time pad.
- Without them, keys are bad, clever algorithms are worthless.

The random (cont.)

Some sources:

- noisy diodes,
- air turbulence in computer disks,
- moving human users,
- lava lamps.

Such sources may be slow, may yield some patterns.

Typically, one stretches and spreads around the randomness (e.g., through one-way hash functions).

Lava lamps



Some theory

Defining (shared-key) encryption

An encryption scheme consists of algorithms:

$$\mathcal{K} : \text{Parameter} \times \text{Coins} \rightarrow \text{Key}$$

$$\mathcal{E} : \text{Key} \times \text{String} \times \text{Coins} \rightarrow \text{Ciphertext}$$

$$\mathcal{D} : \text{Key} \times \text{String} \rightarrow \text{Plaintext}$$

where $\text{Parameter} = 1^*$ (numbers in unary),
Key, Plaintext, Ciphertext \subseteq String.

For all $\eta \in \text{Parameter}$, $k \in \mathcal{K}(\eta)$, and $r \in \text{Coins}$,

- if $m \in \text{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m, r)) = m$,
- if $m \notin \text{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m, r)) = \mathbf{0}$

where $\mathbf{0} \in \text{Plaintext}$ (a fixed string).

Defining secure encryption (1)

Idea:

An adversary cannot recover the key k from $\mathcal{E}_k(m, r)$ (with or without knowledge of m and r).

Immediate objection:

The definition may be a little too strong:

The adversary might succeed only with low probability or after a lot of work.

(And we will need to take probabilities into account.)

Another objection:

The definition is satisfied by a scheme where keys are very strong but where $\mathcal{E}_k(m, r) = m$.

So the first idea is pretty worthless.

We will need to focus more on plaintexts and ciphertexts.

Defining secure encryption (2)

Idea:

Encryptions look random.

Objection:

An encryption scheme that starts every ciphertext with a 0 may be perfectly fine, even if encryptions don't look random.

Thus, randomness of ciphertexts is not so relevant.

Defining secure encryption (3)

Idea:

An adversary cannot recover the plaintext m from $\mathcal{E}_k(m, r)$ (without knowledge of k , but with or without knowledge of r).

Objection:

But the adversary may still be able to gather a lot of sensitive information about m , for example some part of m .

So the definition is too weak.

Defining secure encryption (4)

Idea:

Whatever is efficiently computable from the ciphertext is also efficiently computable without it.

Objection:

But what about the ciphertext itself, and the length of the plaintext?

Still, this idea can be worked out rigorously and yields a good definition.

Defining secure encryption (5)

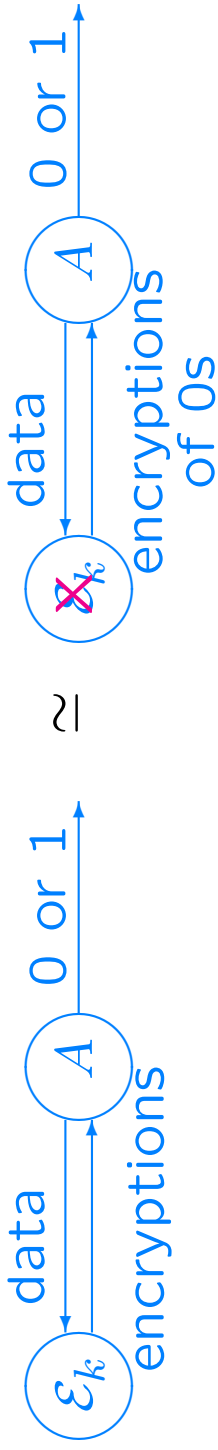
Idea:

It is hard to distinguish a real encryption “box” from a fake one.

This idea is the one that we will explore further.

Secure encryption

An encryption scheme is secure if it is hard to distinguish a real encryption “box” from a fake one.



Negligible

A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}$ is *negligible* if for all $c > 0$ there exists N such that $\epsilon(\eta) \leq \eta^{-c}$ for all $\eta \geq N$.

Secure encryption: definition

Encryption scheme Π is secure if, for every polynomial-time adversary A ,

$$\text{Adv}_{\Pi[\eta]}(A) \triangleq \Pr \left[k \stackrel{R}{\leftarrow} \mathcal{K}(\eta) : A^{\mathcal{E}_k(\cdot)}(\eta) = 1 \right] - \Pr \left[k \stackrel{R}{\leftarrow} \mathcal{K}(\eta) : A^{\mathcal{E}_k(0^{|l})}(\eta) = 1 \right]$$

is negligible.

$\text{Adv}_{\Pi[\eta]}(A)$ is the **advantage** of adversary A .

The “box” is an oracle: a function that the adversary can invoke at will.

- $\mathcal{E}_k(\cdot)$ encrypts its argument.
- $\mathcal{E}_k(0^{|l|})$ encrypts a string of 0's of the same length as its argument.

Some support for the definition

This definition might be counterintuitive.

(E.g., why are the plaintexts in clear?)

Still:

- it is equivalent to other definitions;
- it implies expected properties.

Example

Suppose that there is a function f that recovers the first bit of m from $\mathcal{E}_k(m, r)$.

Then we can construct an adversary A as follows:

- A calls its oracle with a string of 1's;
- A runs f on the output;
- A returns the result;
- $\text{Adv}_{\Pi[\eta]}(A) = 1$

which is not negligible.

(Similarly, $\text{Adv}_{\Pi[\eta]}(A)$ is not negligible even if f recovers the first bit of m only much of the time.)

Deterministic encryption

By this definition, a deterministic encryption scheme cannot be secure.

The following adversary A gets a high advantage:

- A calls its oracle twice, with two different plaintexts of the same length;
- A returns 1 if the results are different.

In essence, deterministic encryption can leak plaintext equalities.

Going further

Nothing here says that:

- the output of encryption should “look random” ,
- encryption provides integrity,
- one can tell whether one used the “right” key for decrypting,
- encryption hides plaintext size,
- ∴

Some common assumptions and some stronger definitions do.

Going further (cont.)

This approach is not specific to shared-key encryption.

It applies to other cryptographic operations as well.

It yields precise definitions and some proofs.