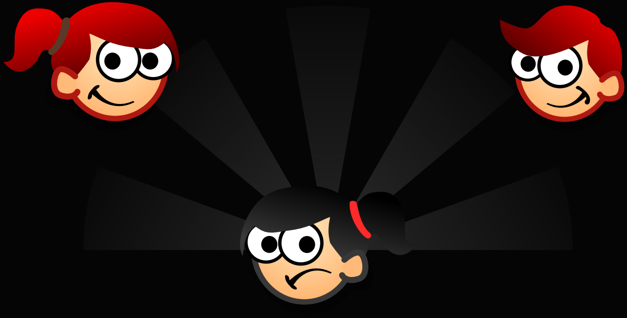


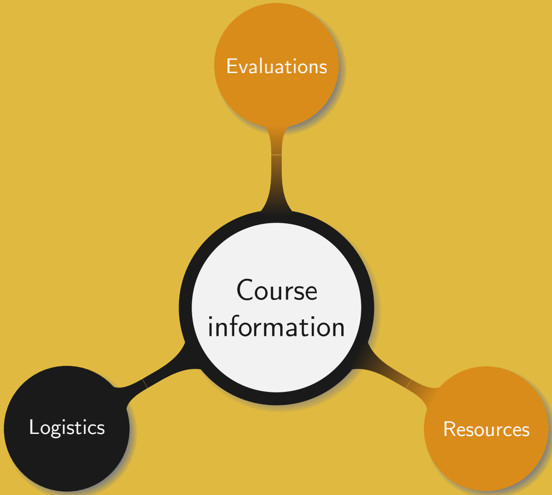
# Introduction to Cryptography

Manuel – Summer 2021

## Table of contents

## 0. Course information

## Chapter organisation



## Basic information

Teaching team:

- Instructor: Manuel (charlem@sjtu.edu.cn)
- Teaching assistants:
  - Hanyu (whynull@sjtu.edu.cn)
  - Yichao (wangyichao1304@sjtu.edu.cn)

Important rules:

- When contacting a TA for an important matter, `CC` the instructor
- Add the tag [VE475] to the subject, e.g. `Subject: [VE475] Grades`
- Use SJTU jBox service to share large files ($>$ 2 MB)

**Never send large files by email**

## Course schedule

Course arrangements:

- Lectures:
    - Tuesday 14:00 – 15:40
    - Thursday 14:00 – 15:40
    - Friday 12:10 – 13:50 (even weeks)
- Office hours:
    - Anytime (Piazza)
    - On appointment (Zoom)

## Course objectives

Primary goals:

- Understand the basics of cryptology and security
- Become familiar with the most common cryptographic protocols
- Be able to relate theory and practice in cryptology

*Decide on the validity and security of given cryptographic solutions*

## Course workflow

Learning strategy:

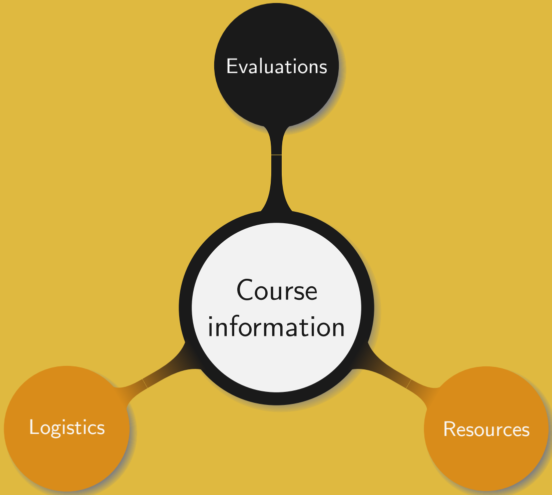
- Course side:
  1. Understand the basic concepts of cryptography
  2. Know the most common problems and their solutions
  3. Get an overview of many subfields of cryptography
- Personal side:
  1. Perform extra research
  2. Relate known strategies to new problems
  3. Read and write some code

## Course outcomes

Detailed goals:

- Know the most common symmetric key cryptography protocols
- Know the most common public key cryptography protocols
- Understand the importance of true randomness in cryptography
- Understand the basics on hash functions in cryptography
- Know the various security levels and be able to derive their corresponding key length depending on the most efficient attacks available
- Know the basic algorithms to solve real life problems such as digital signatures, secret sharing, or traitor tracing
- Be able to perform basic programming in a cryptographic context, i.e. using large numbers or low level logical operations
- Get a high level overview of the various sub-fields of cryptography
- Understand the mathematics used in cryptography

## Chapter organisation



## Assignments

Homework:

- Total: 10
- Content: basic concepts, coding, mathematics

Projects:

- Total: 2
- Content: discover new areas of cryptology

Challenges:

- Total: 3
- Content: code breaking

## Grading policy

Grade weighting:

- Homework: 15%
- Projects: 25%
- Final exam: 30%
- Midterm exam: 30%

Assignment submissions:

- Bonus: $+10\%$ for a work fully written in LaTeX, limited to 100%
- Penalty: $-10\%$ for a work not written in a neat and legible fashion
- Late policy: $-10\%$ per day, not accepted after three days

*Grades will be curved with the median in the range* $[\![B, B+]\!]$

## Honor code

General rules:

- Not allowed:
  - Reuse the code or work from other students or groups
  - Reuse the code or work from the internet
  - Share too many details on how to complete a task
- Allowed:
  - Reuse part the course or textbooks and quoting the source
  - Share ideas and understandings on the course
  - Provide hints on where or how to find information

## Honor Code

Documents allowed during the exams:

- Part A: a mono or bilingual dictionary
- Part B:
    - The lecture slides with **notes on them** (paper or electronic)
    - A mono or bilingual dictionary

Group works:

- Every student in a group is responsible for his group's submission
- If a student breaks the Honor Code, the whole group is guilty

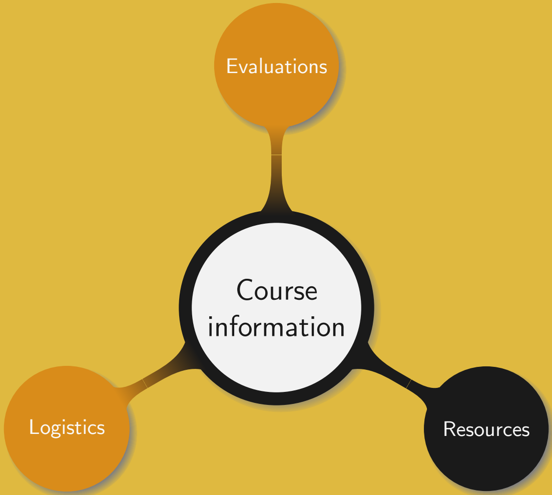
## Special circumstances

Contact us as early as possible when:

- Facing special circumstances, e.g. full time work, illness
- Feeling late in the course
- Feeling to work hard without any result

**Any late request will be rejected**

## Chapter organisation



## Canvas

Information and documents available on the Canvas platform:

- Course materials:
  - Syllabus
  - Lecture slides
  - Homework
  - Projects
  - Challenges
- Course information:
  - Announcements
  - Notifications
  - Grades
  - Polls

## References

Useful places where to find information:

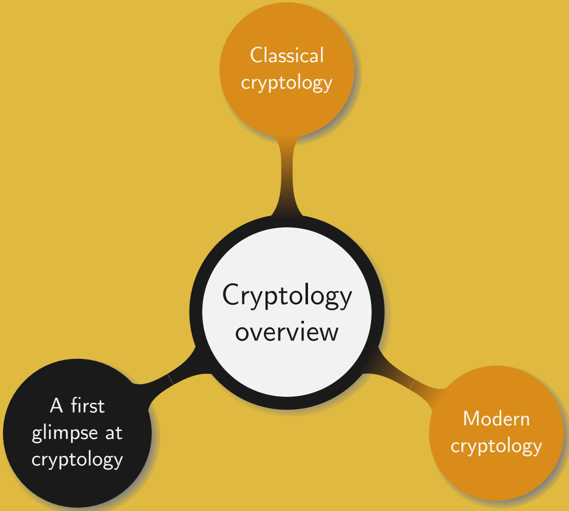
- *Introduction to Modern Cryptography* (J. Katz and Y. Lindell)
- *Cryptography, theory and practice* (D. Stinson)
- Search information online, i.e. $\{\textit{websites} \setminus \{\textit{non-English websites}\}\}$

## Key points

- Work regularly, do not wait the last minute/day
- Respect the Honor Code
- Go beyond what is taught
- Do not learn, understand
- Keep in touch with us
- Advice and suggestions are always much appreciated

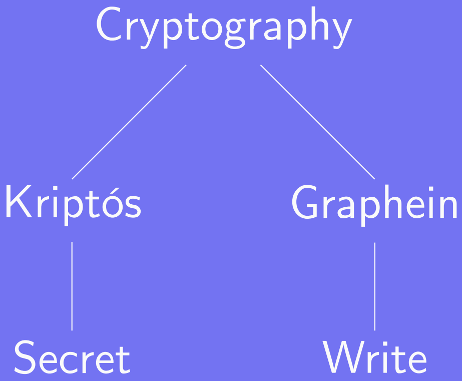
## 1. Cryptology overview

## Chapter organisation

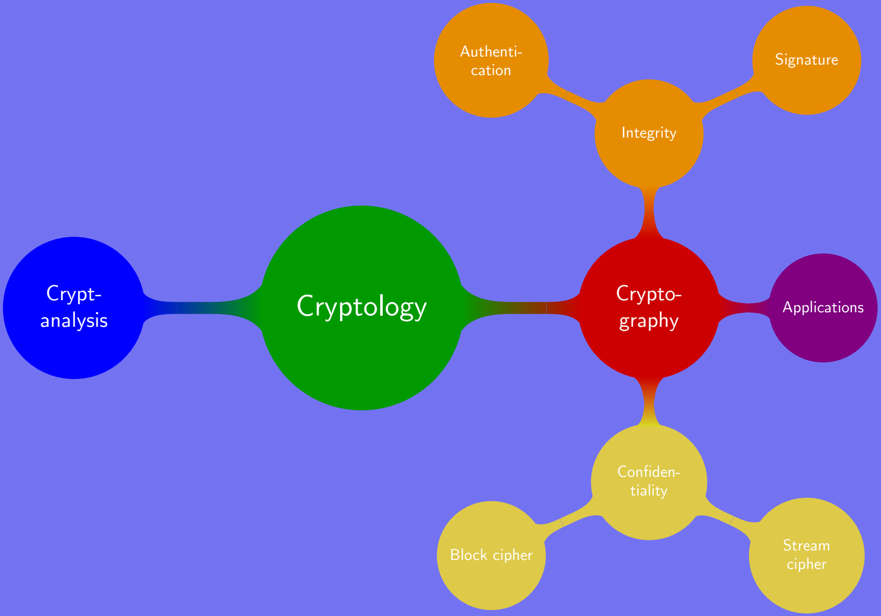


Are you following the right course?

## Etymology



## Cryptology


Crypt-analysis
Cryptology
Crypto-graphy
Integrity
Authenti-cation
Signature
Applications
Confiden-tiality
Block cipher
Stream cipher

## A brief history of cryptology – Early times

Cryptology as an old science:

| | |
|---|---|
| -1500 | Mesopotamia: earliest known use |
| -700 | Greeks: military applications |
| -500 | Common among Hebrew scholars |
| -100 | Romans |
| 800 | Arabs: document cryptanalysis methods |

*Advantage was on the side of cryptanalysts*

## A brief history of cryptology – Until World War II

No major advances until World War I:

| | |
|---|---|
| 1914 | Basic and insecure encryption methods |
| 1917 | Discovery of an unbreakable cipher |
| 1920 | Development of electromechanical devices |
| 1929 | First usage of mathematics |
| 1939 | Major breakthroughs |

*Advantage is still on the side of cryptanalysts*

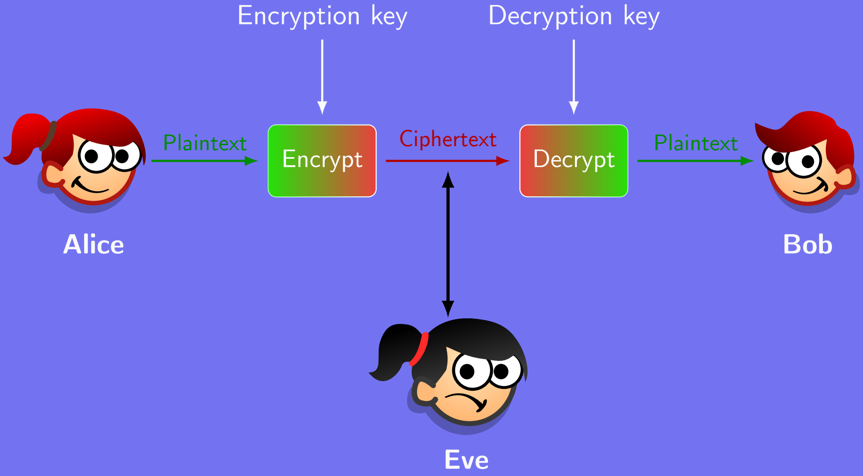
## A brief history of cryptology – Modern times

Mathematics becomes the heart of cryptography:

| | |
|---|---|
| 1949 | Shannon: secrecy et authenticity |
| 1976 | Diffie et Hellman: public key cryptography |
| 1990 | Internet popularize the use of cryptography |
| 2017 | Cryptology is everywhere |

*Politics try to kill cryptography and give governments the monopoly*

## Basic communication scenario



## Eve's goals

Eve has one of the following goals:

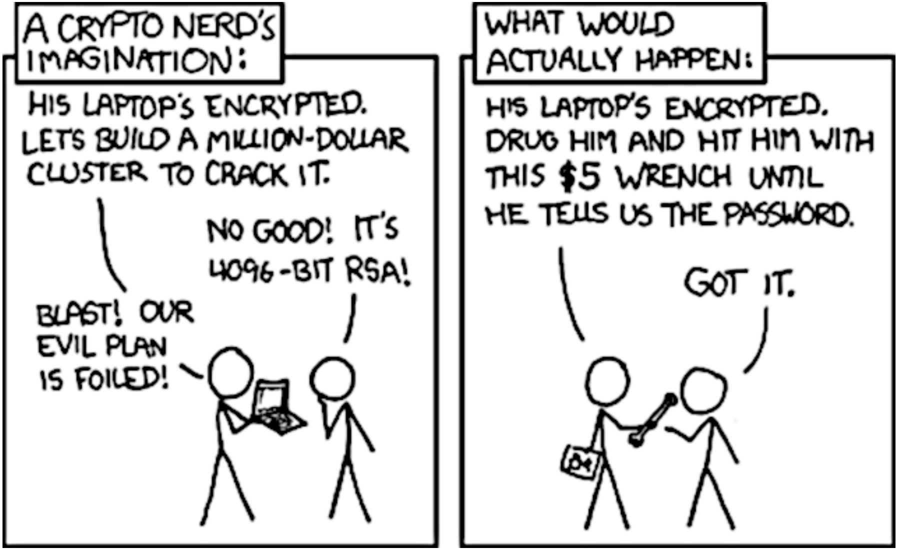
- Read a message
- Find the key
- Corrupt Alice's message
- Masquerade as Alice

## Eve's strategy

There are the five main types of attacks:

- Eve only has a copy of the ciphertext: *ciphertext only*
- Eve has a copy of the ciphertext but also of the corresponding plain-text: *Known Plaintext Attack (KPA)*
- Eve chooses the plaintext to be encrypted: *Chosen Plaintext Attack (CPA)*
- Eve chooses the ciphertext to be decrypted: *Chosen Ciphertext Attack (CCA)*
- Eve chooses any plaintext to be encrypted or ciphertext to be decryp-ted: *Chosen Plaintext and Ciphertext Attack (CPCA)*

## Eve's strategy


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LETS BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

## Eve’s strategies

Methods to collect data:

- On fiber cables and infrastructures as the flow passes
- From the servers of service providers

Methods to retrieve encrypted data:

- Break the encryption
- Influence industrial standards
- Pressure manufacturers to make insecure devices
- Infiltrate hardware and software

## Who is Eve and why is she evil?

Eve is anyone that might want to read or temper the data:

- Low threat: friends, family members, etc.
- High threat: governmental agencies and companies

Reasons for mass surveillance:

- Combat terrorism
- Assess foreign policies and economical stability
- Gather commercial secrets

## Why fear Eve?

What does your phone know about you?

*"They (the NSA) can use the system to go back in time and scrutinize every decision you've ever made, every friend you've ever discussed something with, and attack you on that basis to sort of derive suspicion from an innocent life and paint anyone in the context of a wrongdoer."*

Edward Snowden

## Kerckhoffs' principle

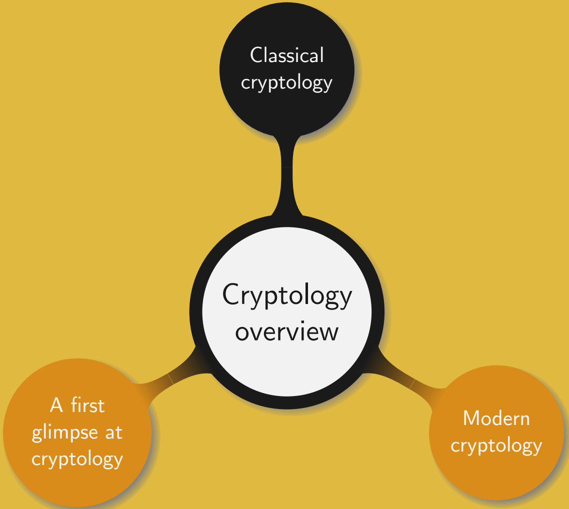
**Principle** (Kerckhoffs' principle)

A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

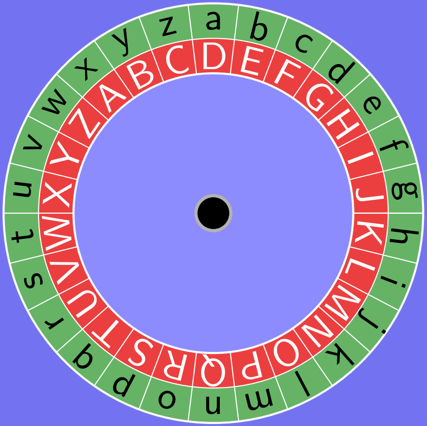
In other words:

- Security through obscurity is not security
- Data should be encrypted using standard, publicly known algorithms
- The implementation must be accessible to all

## Chapter organisation



## Caesar 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
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

Simple description:

- One of the earliest cipher
- Attributed to Julius Caesar
- Letters are shifted by a given number of places
- The shift is called the *key* of the cipher

Exercise. Recover the plaintext given the ciphertext JRRGPRUQLQJ

## Modular arithmetic

**Definitions**

1. Let $a$ and $b$ be two integers, with $a \neq 0$. We say that *$a$ divides $b$* if there exists an integer $k$ such that $b = ak$, and we denote it $a|b$.
2. Let $a$, $b$ and $n$ be three integers with $n \neq 0$. We say that *$a$ is congruent to $b$ modulo $n$*, if $n$ divides $a - b$. It is denoted $a \equiv b \bmod n$

In modern cryptography:

- The plaintext is first converted into a numerical value
- If the alphabet is composed of $n$ symbols then each one is assigned a value between 0 and $n - 1$

## Revisiting Caesar cipher

Caesar cipher in mathematical terms:

1. Label letters as integers from 0 to 25
2. Choose a key $\kappa$ in the range 0 – 25
3. Encrypt using the function $x \mapsto x + \kappa \bmod 26$
4. Decrypt using the function $x \mapsto x - \kappa \bmod 26$
5. Label integers from 0 to 25 as letters

Exercise. Encrypt and decrypt “students are working hard” using Caesar cipher with the key $\kappa = -5$

## Breaking Caesar cipher

Using the different types of attacks:

- Ciphertext only: only 26 possible keys $\Rightarrow$ exhaustive search
- KPA: deduce the key from the plaintext/ciphertext pair
- CPA: for the plaintext "a", the ciphertext gives $\kappa$
- CCA: for the ciphertext "A", the plaintext gives $-\kappa \bmod 26$

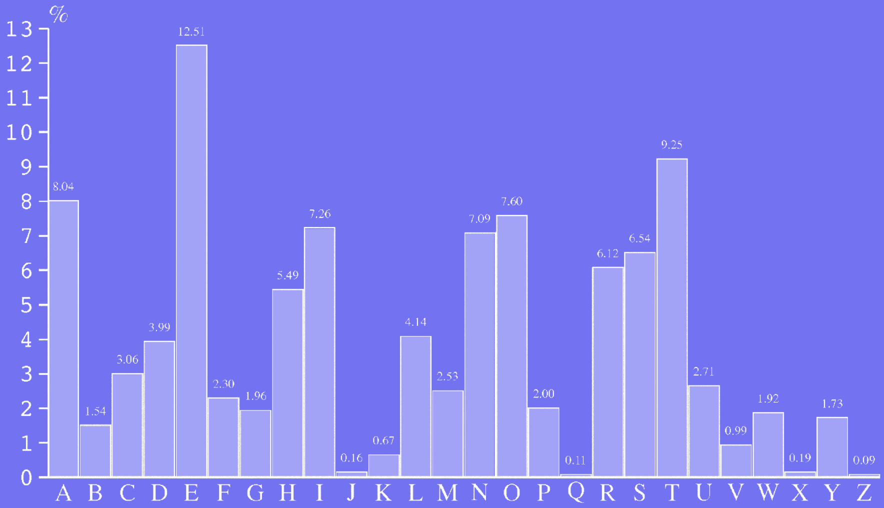
## Substitution ciphers

In 1776 Thomas Jefferson sent a message to Benjamin Franklin:

LWNSOZBNWVWBAYBNVBSQWVUOHWDIZWRBBNPBPOOUWRPAWX
AWPBWZWMYPOBNPBBNWJPAWWRZSLWZQJBNVIAXAWPBSALIBNX
WABPIRYRPOIWRPQOWAIENBVBNPBPUSREBNWVWPAWOIHWOIQW
ABJPRZBNWFYAVYIBSHNPFFIRWVVBNPBBSVWXYAWBNWVWAIENB
VESDWARUWRBVPAWIRVBIBYBWZPUSREUWRZWAIDIREBHWIATYVB
FSLWAVHASUBNWXSRVWRBSHBOTESDWARWZBNPBLNWWDWAPRJ
HSAUSHESDWARUWRBQWXSUWVZWVBAYXBIDWSHBNWVWWRZVIB
IVBNVAIENBSHBNWFWSFOWBSPOBWASABSPQSOIVNIBPRZBSIRVBI
BYBWRWLESDWARUWRBOPJIREIBVHSYRZPBISRSRVYXNFAIRXIFOO
TPRZSAEPRIKIREIBVFSLWAVIRVYXNHSAUPVBSVWMJSVBOICWOJBS
WHHWXBBNWIAVPHWBJPRZNPFFIRWW

*Your job is to decipher it*

## Letters distribution in English



## Decrypting the message – Monograms

For the 10 most common letters their count gives:

| W | B | R | S | I | V | A | P | N | O |
|---|---|---|---|---|---|---|---|---|---|
| 76 | 64 | 39 | 36 | 36 | 35 | 34 | 32 | 30 | 16 |

We can guess:

- W is probably e
- B, R, S, I, V, A, P, and N are probably t, a, o, i, n, s, h, and r
- But what is their order?

## Decrypting the message – Digrams

### Digrams count

| | W | B | R | S | I | V | A | P | N |
|---|---|---|---|---|---|---|---|---|---|
| W | 3 | 4 | 12 | 2 | 4 | 10 | 14 | 3 | 1 |
| B | 4 | 4 | 0 | 11 | 5 | 5 | 2 | 4 | 20 |
| R | 5 | 5 | 0 | 1 | 1 | 5 | 0 | 3 | 0 |
| S | 1 | 0 | 5 | 0 | 1 | 3 | 5 | 2 | 0 |
| I | 1 | 8 | 10 | 1 | 0 | 2 | 3 | 0 | 0 |
| V | 8 | 10 | 0 | 0 | 2 | 2 | 0 | 3 | 1 |
| A | 7 | 3 | 4 | 2 | 5 | 4 | 0 | 1 | 0 |
| P | 0 | 8 | 6 | 0 | 1 | 1 | 4 | 0 | 0 |
| N | 14 | 3 | 0 | 1 | 1 | 1 | 0 | 7 | 0 |

### Rules in English

- e contacts most of other letters
- a, i, o tend to avoid each other
- 80% of the letters preceding n are vowels
- the most common digram is th
- h often appears before e, rarely after
- r pairs more with vowels and s with consonants
- rn more common than nr and to than ot

## Completing the decryption

Summarizing all the guesses and carrying on:

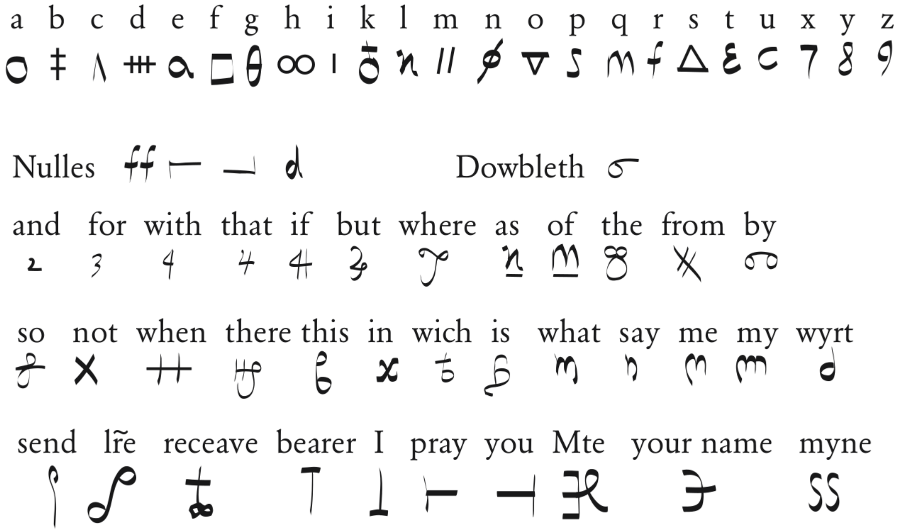
| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L | W | N | S | O | Z | B | N | W | V | W | B | A | Y |
| **w** | **e** | **h** | **o** | **l** | **d** | t | h | e | s | e | t | r | u |
| B | N | V | B | S | Q | W | V | W | O | H | W | D | **I** |
| t | h | s | t | o | **b** | e | s | e | l | f | e | v | i |
| Z | W | R | B | B | N | P | B | P | … | | | | |
| d | e | n | t | t | h | a | t | a | | | | | |

## Deciphered text

The deciphered text is from the Declaration of independence:

we hold these truths to be self evident that all men are created equal that they are endowed by their creator with certain unalienable rights that among these are life liberty and the pursuit of happiness that to secure these rights governments are instituted among men deriving their just powers from the consent of the governed that whenever any form of government becomes destructive of these ends it is the right of the people to alter or to abolish it and to institute new government laying its foundation on such principles and organizing its powers in such form as to seem most likely to effect their safety and happiness

## Mary, the queen of Scots


a b c d e f g h i k l m n o p q r s t u x y z
Nulles
Dowbleth
and for with that if but where as of the from by
so not when there this in wich is what say me my wyrt
send lre receave bearer I pray you Mte your name myne

## One Time Pad

Using the One Time Pad:

1. Represent the message as a sequence of 0s and 1s of length $l$
2. Generate a key of length $l$ and composed of 0s and 1s
3. XOR the message and the key

Breaking the One Time Pad:

- Ciphertext only: all the messages of same length have equal probability
- KPA, CPA, CCA: only reveal part of the key used during the attack

## Hill cipher

A *block cipher* encrypts several letters at once:

- Changing one letter in the plaintext impacts several letters in the ciphertext
- Frequency analysis of letters and digrams cannot be applied

Hill cipher:

- Invented in 1929
- One of the first cipher to use algebraic methods
- Never been used much in practice

## Algebraic digression – Greatest common divisor

**Definition**

The *greatest common divisor* of two integers $a$ and $b$, with $|a| + |b| \neq 0$, is the largest positive integer dividing both $a$ and $b$. It is noted $\gcd(a, b)$, and $a$ and $b$ are said to be *coprime* if $\gcd(a, b) = 1$.

In fact $\gcd(a, b)$ can be expressed as a linear combination of $a$ and $b$ with integer coefficients.

**Lemma** (Bézout's identity)

Let $a$ and $b$ be two integers where at least one of them is not zero, and $d = \gcd(a, b)$. Then there exists two integers $s$ and $t$, called *Bézout coefficients*, such that $as + bt = d$.

## Algebraic digression – Computing the gcd

Algorithm. (*Extended Euclidean Algorithm*)

```
Input  : $a$, $b$, two positive integers
Output: $r_1 = \gcd(a, b)$ and $\langle s_1, t_1 \rangle$, Bézout coefficients
1  $r_0 \leftarrow b$; $r_1 \leftarrow a$;
2  $s_0 \leftarrow 0$; $s_1 \leftarrow 1$;
3  $t_0 \leftarrow 1$; $t_1 \leftarrow 0$;
4  while $r \neq 0$ do
5      $q \leftarrow r_1 \operatorname{div} r_0$;
6      $\langle r_1, r_0 \rangle \leftarrow \langle r_0, r_1 - qr_0 \rangle$;
7      $\langle s_1, s_0 \rangle \leftarrow \langle s_0, s_1 - qs_0 \rangle$;
8      $\langle t_1, t_0 \rangle \leftarrow \langle t_0, t_1 - qt_0 \rangle$;
9  end while
10 return $r_1$, $\langle s_1, t_1 \rangle$
```

## Algebraic digression – Multiplicative inverse

**Proposition**

Let $a$ and $n$ be two coprime integers and $s$ and $t$ be such that $as+nt=1$. Then $as \equiv 1 \bmod n$, and $s$ is called the *multiplicative inverse* of $a$ modulo $n$. Besides $s$ is unique.

Example. What is the multiplicative inverse of 11111 modulo 12345?

Running the extended Euclidean algorithm confirms that 11111 and 12345 are coprime and therefore 11111 is invertible modulo 12345. Moreover since

$$11111 \cdot 2471 + 12345 \cdot (-2224) = 1,$$

we conclude that $11111 \cdot 2471 \equiv 1 \bmod 12345$.

## Algebraic digression – Matrix inversion

**Theorem** (Cramer's rule)

Let $A$ be an $m \times m$ matrix, then

$$\mathrm{Adj}(A) \cdot A = \det(A)\, \mathrm{I}_m, \tag{1.1}$$

where $\mathrm{Adj}(A)$ denotes the adjugate of $A$, $\det(A)$ the determinant of $A$, and $\mathrm{I}_m$ the $m \times m$ identity matrix.

From equation (1.1) we see that for $A$ to be invertible, $\det(A)$ must be invertible. In particular if $A$ is defined modulo $n$, $\det(A)$ must be invertible modulo $n$, that is there exists $t$ such that

$$\det(A) \cdot t \equiv 1 \bmod n.$$

## Algebraic digression – Modular matrix inversion

Example. Compute the inverse of the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{pmatrix} \bmod 11.$$

Since $\det(A) = 2$ and $\gcd(2, 11) = 1$, $A$ is invertible modulo 11 and

$$A^{-1} = \frac{1}{2} \begin{pmatrix} +\begin{vmatrix} 2 & 3 \\ 4 & 9 \end{vmatrix} & -\begin{vmatrix} 1 & 1 \\ 4 & 9 \end{vmatrix} & +\begin{vmatrix} 1 & 1 \\ 2 & 3 \end{vmatrix} \\ -\begin{vmatrix} 1 & 3 \\ 1 & 9 \end{vmatrix} & +\begin{vmatrix} 1 & 1 \\ 1 & 9 \end{vmatrix} & -\begin{vmatrix} 1 & 1 \\ 1 & 3 \end{vmatrix} \\ +\begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} & -\begin{vmatrix} 1 & 1 \\ 1 & 4 \end{vmatrix} & +\begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix} \end{pmatrix} \bmod 11.$$

## Algebraic digression

Then calculating all the cofactors yields

$$A^{-1} = \frac{1}{2}\begin{pmatrix} 6 & -5 & 1 \\ -6 & 8 & -2 \\ 2 & -3 & 1 \end{pmatrix} \bmod 11.$$

In this case it is easy to see that 6 is the inverse of 2 modulo 11, such that we get

$$A^{-1} = \begin{pmatrix} 36 & -30 & 6 \\ -36 & 48 & -12 \\ 12 & -18 & 6 \end{pmatrix} \equiv \begin{pmatrix} 3 & 3 & 6 \\ 8 & 4 & 10 \\ 1 & 4 & 6 \end{pmatrix} \bmod 11.$$

## Back to Hill cipher

Constructing Hill cipher:

- Key: generate a random $n \times n$ matrix $K$ modulo 26, such that $\gcd(\det(K), 26) = 1$
- Encrypt:
    - Split the plaintext into blocks of size $n$, padding with extra letters if necessary
    - Multiply each block considered as a vector by the matrix $K$
- Decrypt:
    - Split the ciphertext into blocks of size $n$
    - Multiply each block considered as a vector by the matrix $K^{-1}$

## Hill cipher

Example. Encrypt "good morning" with the key $K = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 9 & 7 & 8 \end{pmatrix}$.

1. Split and pad the plaintext

$$\underbrace{\begin{matrix} \mathrm{g} & \mathrm{o} & \mathrm{o} \\ 6 & 14 & 14 \end{matrix}}_{A} \; \underbrace{\begin{matrix} \mathrm{d} & \mathrm{m} & \mathrm{o} \\ 3 & 12 & 14 \end{matrix}}_{B} \; \underbrace{\begin{matrix} \mathrm{r} & \mathrm{n} & \mathrm{i} \\ 17 & 13 & 8 \end{matrix}}_{C} \; \underbrace{\begin{matrix} \mathrm{n} & \mathrm{g} & \mathrm{x} \\ 13 & 6 & 23 \end{matrix}}_{D}$$

2. Multiply each vector by $K$

$$\overbrace{\begin{matrix} 6 & 24 & 6 \\ \mathrm{G} & \mathrm{Y} & \mathrm{G} \end{matrix}}^{A'} \; \overbrace{\begin{matrix} 21 & 8 & 11 \\ \mathrm{V} & \mathrm{I} & \mathrm{L} \end{matrix}}^{B'} \; \overbrace{\begin{matrix} 11 & 25 & 11 \\ \mathrm{L} & \mathrm{Z} & \mathrm{L} \end{matrix}}^{C'} \; \overbrace{\begin{matrix} 10 & 9 & 25 \\ \mathrm{K} & \mathrm{J} & \mathrm{Z} \end{matrix}}^{D'}$$

## KPA on Hill cipher

Knowing “goodmorningx” and “GYGVILLZLKJZ” recover the key.

1. Find $n$: since $n|12$, try some values until the right one is found
2. Use the three first blocks to construct the equation

$$\underbrace{\begin{pmatrix} 6 & 14 & 14 \\ 3 & 12 & 14 \\ 17 & 13 & 8 \end{pmatrix}}_{A} \cdot \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \equiv \begin{pmatrix} 6 & 24 & 6 \\ 21 & 8 & 11 \\ 11 & 25 & 11 \end{pmatrix} \bmod 26$$

3. Since $A$ is not invertible modulo 26, try with the three last blocks

$$\underbrace{\begin{pmatrix} 3 & 12 & 14 \\ 17 & 13 & 8 \\ 13 & 6 & 23 \end{pmatrix}}_{A} \cdot \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \equiv \begin{pmatrix} 21 & 8 & 11 \\ 11 & 25 & 11 \\ 10 & 9 & 25 \end{pmatrix} \bmod 26$$

## KPA on Hill cipher

4. Since $A$ is now invertible we calculate

$$K = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \equiv \begin{pmatrix} 3 & 12 & 14 \\ 17 & 13 & 8 \\ 13 & 6 & 23 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 21 & 8 & 11 \\ 11 & 25 & 11 \\ 10 & 9 & 25 \end{pmatrix} \mod 26$$

$$K = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \equiv \begin{pmatrix} 11 & 18 & 4 \\ 7 & 11 & 10 \\ 1 & 22 & 11 \end{pmatrix} \cdot \begin{pmatrix} 21 & 8 & 11 \\ 11 & 25 & 11 \\ 10 & 9 & 25 \end{pmatrix} \mod 26$$

And the key is

$$K = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 9 & 7 & 8 \end{pmatrix}.$$

## Hill cipher

Remarks on Hill cipher:

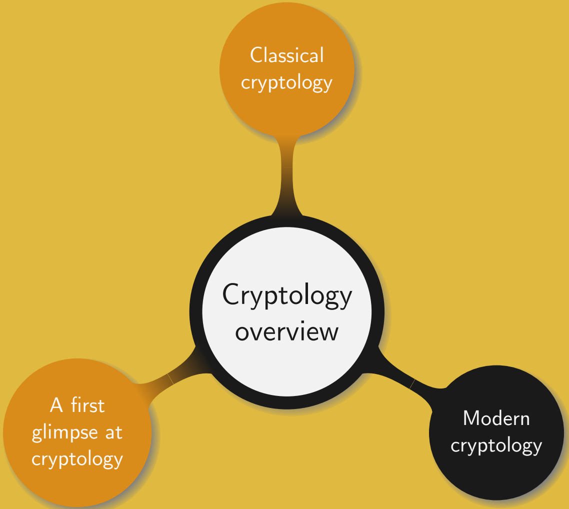
- In a substitution cipher, changing one letter from the plaintext alters one letter from the ciphertext
- In Hill cipher changing one letter from the plaintext alters the whole corresponding block from the ciphertext
- Hill cipher is not vulnerable to frequency analysis attacks
- As a drawback a small error in the transmission can induce a major error in the encrypted message and the deciphered text becomes unreadable

## The Enigma machine

Device information:

- Developed in Germany during the 1920s
- 1054560 ways to initialise the machine
- 100391791500 ways to interchange six pairs of letters
- Secretly broken in Poland in the 1930s
- Techniques extended by the British during World War II

## Chapter organisation



## Symmetric vs. Asymmetric

All the previous schemes are symmetric:

- The same key is used to both encrypt and decrypt
- The decryption key is easily derived from the encryption key

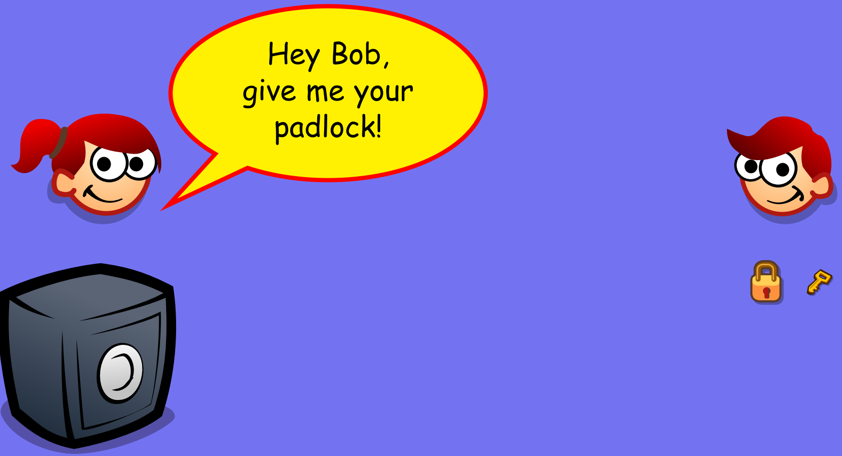
Limitations:

- Alice and Bob need to meet in order to exchange, generate, or share the secret keys
- Key management problem:
    - 2 users $\rightarrow$ 1 key
    - 5 users $\rightarrow$ 4 keys each, total 10 keys
    - $n$ users $\rightarrow$ $n-1$ keys each, total $O(n^2)$ keys

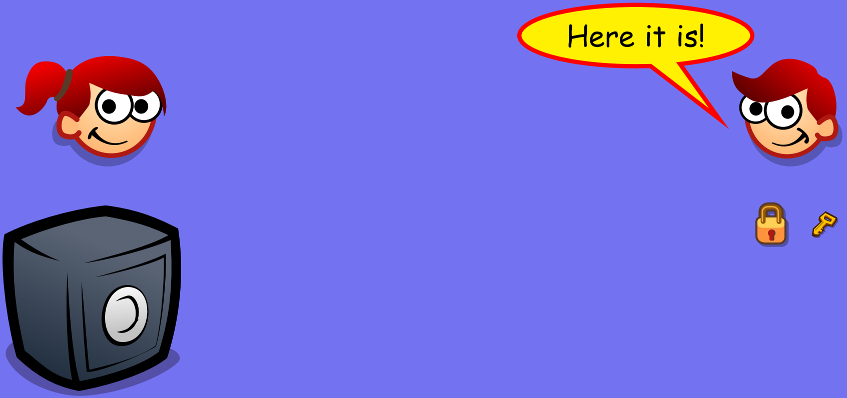
## Public key cryptography

## Public Key Cryptography

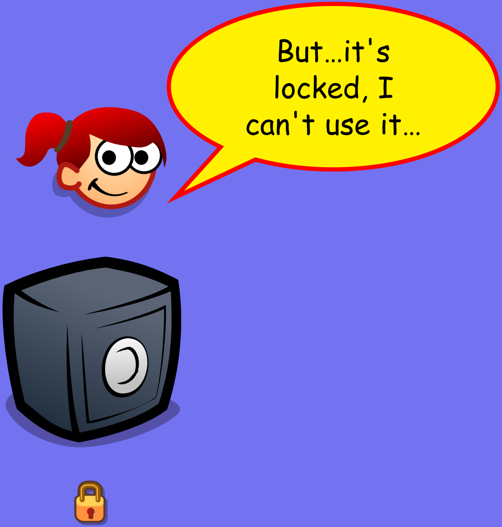
## Public Key Cryptography



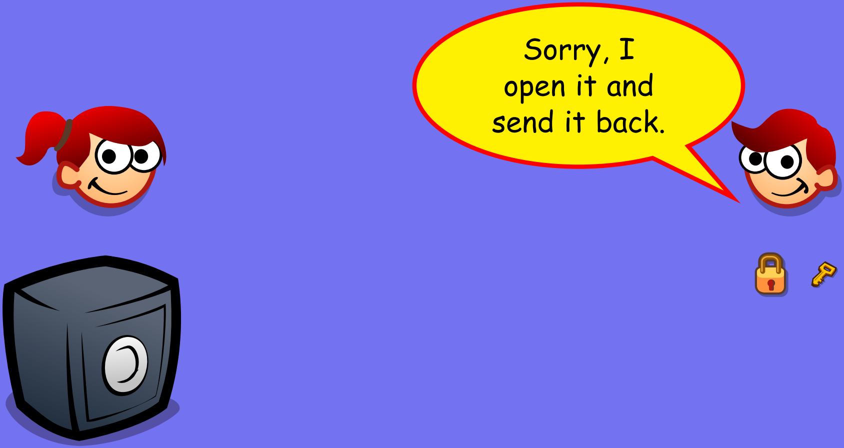
## Public Key Cryptography



## Public Key Cryptography



## Public Key Cryptography



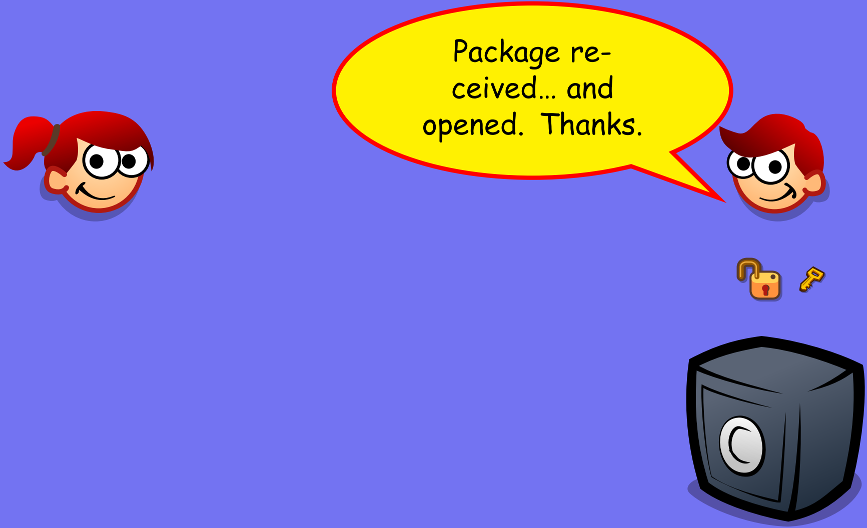
## Public Key Cryptography

## Public Key Cryptography

## Public Key Cryptography

Package ready!

## Public Key Cryptography



## Implementing public key cryptography

*Anybody can lock the padlock but only Bob can unlock it*

Mathematical problems used in Public Key Cryptography (PKC):

- Easy to generate by anybody
- Hard to solve for everybody
- Easy to solve when knowing a small secret

Common examples:

- Multiplication and factorisation
- Exponentiation and discrete logarithm problem

## Historical progression

Over time security has depended on:

- Early years: keeping the encryption method secret
- After WW I: keeping the secret key unknown
- Modern cryptography:
  - The method, the encryption key, and how to find the secret key are known
  - Security depends on the computational infeasibility of finding it

*PKC adds much flexibility at a high computational cost*

## Measuring security – Key space

Basic security feeling:

- Obvious strategy: brute force all possible keys
- Intuition: the larger the key space the harder finding the key

Example. Substitution cipher:

- Key space: $26! \approx 4 \cdot 10^{26} \approx 2^{89}$
- Very simple to break using frequency analysis

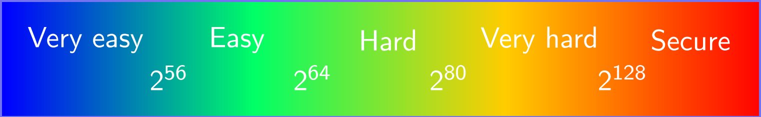
*Brute force is to be used only if no other attack is possible*

## Measuring security – Computational complexity

Best CPUs available in 2015:

- Regular user: 298,190 MIPS (Intel Core i7 5960x)
- Supercomputer: 10,000,000,000 MIPS (Fujitsu K – 705,024 cores)

How many such computers need to run for a year to complete a program composed of $2^{80}$ instructions?



## Complexity and security

*The goal is to be secure in the worst case*

In the worst case the attacker:

- Has the best computational facilities
- Uses the most efficient attack available

To be secure against such an attacker:

- Check to complexity of the best algorithm available
- Adjust the parameters of the cipher such that more than $2^{128}$ operations are required to break the encryption

## Complexity and security

Example. Assuming that the best attack on a mathematical problem requires $\sqrt{n}$ operations, where $n$ is the size of the key, what key size should be chosen to be secure?

Since secure means that the attacker has to compute at least $2^{128}$ operations to break the encryption it suffices to calculate

$$\left(2^{128}\right)^2 = 2^{256}.$$

Hence the key space should contain $2^{256}$ elements, that is the key should be at least 256 bits long.

## Improving security?

*Is double encryption with two different keys enhancing security?*

Improving security:

- Naive answer: for a key of length $k$, $2^{2k}$ operations are needed
- Better answer:
  - It does not change anything, e.g. Hill cipher
  - It is possible to do better than $2^{2k}$: meet in the middle attack

Symmetric encryption using a function $f$ and a key $k$:

- Simple encryption: $c = f_k(m)$
- Double encryption: $c = f_{k_2}(f_{k_1}(m))$
- Decryption: $m = f_{k_1}^{-1}(f_{k_2}^{-1}(c))$

## Meet in the middle attack

Assuming a KPA setup:

1. For all the keys, compute and store the ciphertexts $c_i = f_{k_i}(m)$
2. Compute all plaintexts $m_i = f_{k_i}^{-1}(c)$ and find any matching $c_i$
3. Recover the corresponding keys $k_1$ and $k_2$
4. Test $k_1$ and $k_2$ on more plaintext/ciphertext pairs

Exercise. Assuming no attack applies on an encryption scheme and a key size of 64 bits, what is its security if applying double encryption?

## Complexity theory

Main complexity classes related to cryptology:

- $\mathcal{P}$: decision problems for which there exists a deterministic polynomial time algorithm
- $\mathcal{NP}$: decision problems for which the answer "yes" can be verified using a deterministic polynomial time algorithm
- $\mathcal{NP}$-complete: hardest problems in $\mathcal{NP}$
- co-$\mathcal{NP}$: decision problems for which the answer "no" can be verified using a deterministic polynomial time algorithm
- co-$\mathcal{NP}$-complete: hardest problems in co-$\mathcal{NP}$

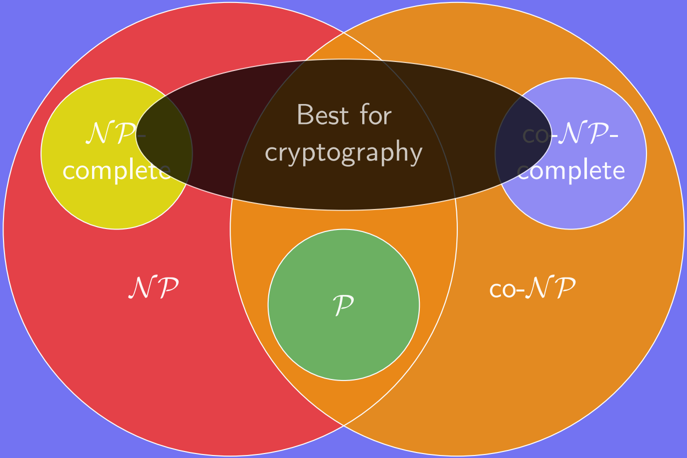
## Complexity theory

Example. Integer factorization is in both $\mathcal{NP}$ and co-$\mathcal{NP}$

Let $n$ be a large integer and $1 < m < n$. Does $n$ have a factor $p$, with $1 < p < m$?

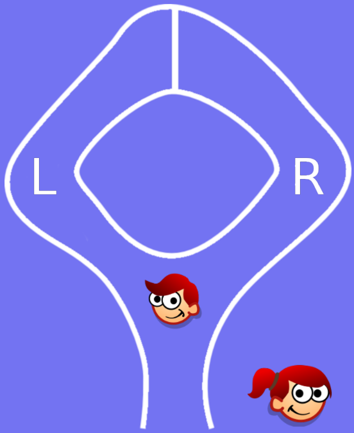
- $\mathcal{NP}$: with certificate "$p$ a factor of $n$" verify in polynomial time that $1 < p < m$ and $p|n$
- co-$\mathcal{NP}$: with certificate "the list of all the prime factors of $n$" verify in polynomial time that:
  - They are all prime
  - Their product is $n$
  - None of them is between 1 and $m$

## Complexity theory



## Zero knowledge proofs

*Bob knows a secret path, and wants to prove it without revealing it*

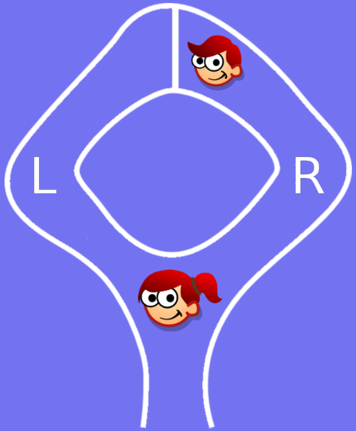


Strategy:

1. Alice hides while Bob chooses to go Left (L) or Right (R)
2. Alice randomly asks Bob to exit on L or R
3. If Bob is on the wrong side he uses the secret path or otherwise returns
4. Repeat steps 1 to 3 many times

## Zero knowledge proofs

*Bob knows a secret path, and wants to prove it without revealing it*

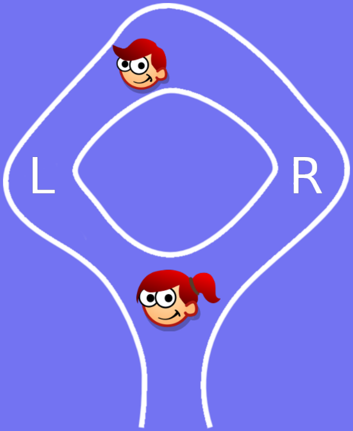


Strategy:

1. Alice hides while Bob chooses to go Left (L) or Right (R)
2. Alice randomly asks Bob to exit on L or R
3. If Bob is on the wrong side he uses the secret path or otherwise returns
4. Repeat steps 1 to 3 many times

## Zero knowledge proofs

*Bob knows a secret path, and wants to prove it without revealing it*



Strategy:

1. Alice hides while Bob chooses to go Left (L) or Right (R)
2. Alice randomly asks Bob to exit on L or R
3. If Bob is on the wrong side he uses the secret path or otherwise returns
4. Repeat steps 1 to 3 many times

## Mathematical digression – Graphs

**Definitions**

1. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two simple graphs. Then we say that $G_1$ and $G_2$ are *isomorphic* if there exists a bijective function $\varphi\colon V_1 \to V_2$ such that the induced map
$$\varphi_*\colon E_1 \to E_2, \qquad (a, b) \mapsto (\varphi(a), \varphi(b))$$
is bijective. Such a function $\varphi$ is called a *graph isomorphism*.
2. A *Hamilton circuit* in a graph $G$ is a simple circuit that passes through every vertex of $G$ exactly once.

## Complexity digression – Graphs

Hard problems related to graph theory:

- Graph isomorphism:
  - No known polynomial time algorithm
  - Not proven to be $\mathcal{NP}$-complete
  - Best known algorithm has exponential complexity
- Finding a Hamiltonian circuit:
  - Proven to be $\mathcal{NP}$-complete
  - Best known algorithm has exponential complexity

## Zero knowledge proofs – Authentication

Initial setup:

- A graph $G$
- A Hamiltonian circuit in $G$

- Bob's graph $G$

Process:

1. Bob generates $H$, a graph isomorphic to $G$
2. Bob commits $H$
3. Alice randomly asks for either the isomorphism or a Hamiltonian circuit in $H$
4. Bob either shows the isomorphism or translates the Hamiltonian circuit in $G$ onto $H$ and shows it

## Zero knowledge proofs

Regarding the process and setup:

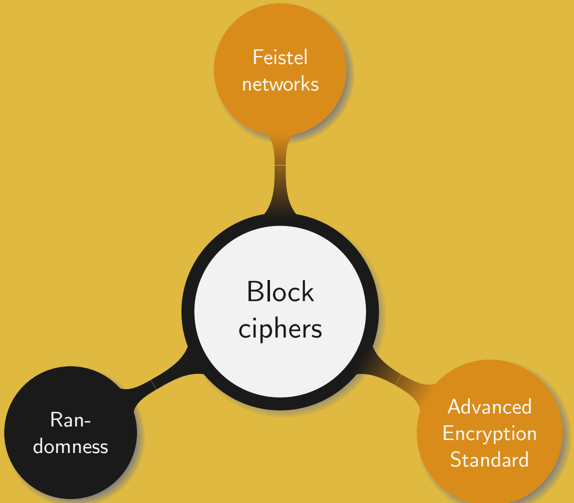
- Is Bob revealing any sensitive information?
- Why does Bob need to commit $H$?
- If Bob can anticipate Alice's request how can he cheat if asked for:
  - The graph isomorphism?
  - The Hamiltonian circuit?
- Explain whether randomness is important or not in the process.
- Can any of Bob and Alice cheat?
- How many times should the process be repeated for Bob to prove that he really knows a Hamiltonian circuit in $G$?

## Key points

- What is cryptology?
- Who are Alice, Bob, and Eve?
- What is Kerckhoff's principle?
- Explain the One-Time-Pad
- Explain the underlying idea of public key cryptography
- In 2021 what security level is considered safe?

## 2. Block ciphers

## Chapter organisation



## Block ciphers

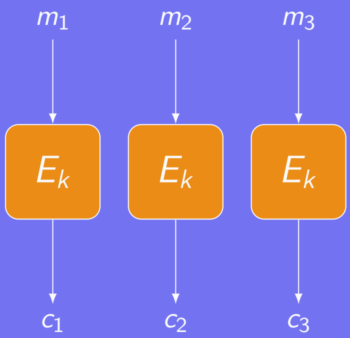
A *block cipher* is composed of two functions, inverse of each other:

$$\begin{aligned} E : \{0,1\}^n \times \{0,1\}^k &\to \{0,1\}^n \\ (P, K) &\mapsto C \end{aligned} \qquad \begin{aligned} D : \{0,1\}^n \times \{0,1\}^k &\to \{0,1\}^n \\ (C, K) &\mapsto P \end{aligned}$$

where $n$ and $k$ are the sizes of a block and the key, respectively.

Goal: given a key $K$, design an invertible function $E$ whose output cannot be distinguished from a random permutation over $\{0,1\}^n$.

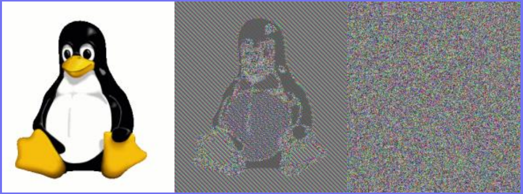
## Mode ECB


$$\begin{tikzcd} m_1 \arrow[d] & m_2 \arrow[d] & m_3 \arrow[d] \\ \boxed{E_k} \arrow[d] & \boxed{E_k} \arrow[d] & \boxed{E_k} \arrow[d] \\ c_1 & c_2 & c_3 \end{tikzcd}$$

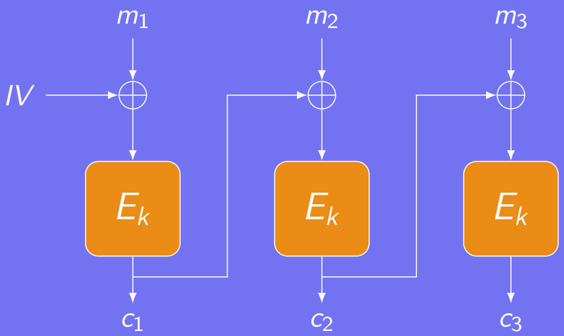
Basic principle:

- Split the plaintext in blocks of size $n$
- Encrypt each block with a function $E$ and a key $K$
- Electronic Code Block (ECB) mode

Limitation: what if a block is repeated several times over the message?



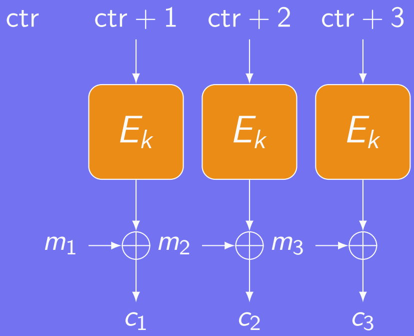
## Mode CBC



Basic principle:

- Cipher Block Chaining (CBC)
- Most commonly used mode
- Uses an Initialization Vector
- Can it be parallelized?

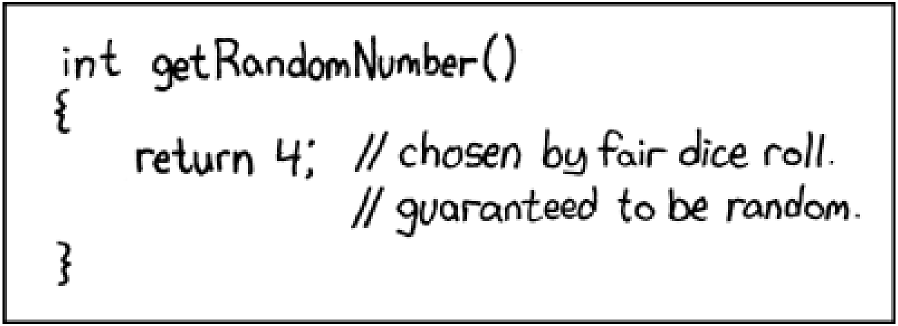
## Mode CTR



Basic principle:

- CTR stands for counter
- The counter acts like an IV
- The $E_K$ function randomizes the counter
- Can be run in parallel

## What is randomness?



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

## Randomness

**Definition** (Kolmogorov randomness)

Let $x$ be a string.

- We say that $x$ is *random* if and only if it is not larger than any program that can produce it in any language.
- The *entropy* of $x$ is the minimum number of bits necessary to describe $x$.

Remark. A random string of length $k$ cannot be compressed in any way, therefore it has entropy $k$

Example. 3434 3434 3434 3434 3434 3434 3434 3434
ed71 b38f 4316 6907 a8ea 75d3 c141 735f

## Random bits generation

Generating true randomness is not simple:

- Toss a coin
- Measure physical phenomena that are expected to be random
- In case of a lack of entropy the output is blocked

Example. The thermal noise from a semiconductor resistor
A nuclear decay radiation source measured by a Geiger counter

## Pseudo-random bits generation

Random function from the C standard:

```
/*  Linear congruential generator */
static unsigned long next = 1;

/* RAND_MAX assumed to be 32767 */
int rand(void) {
  next = next * 1103515245 + 12345;
  return((unsigned)(next/65536) % 32768);
}

void srand(unsigned int seed) {
  next = seed;
}
```

## Pseudo-random bits generation – BBS generator

A secure method from Blum, Blum and Shub:

1. Generate two large primes $p$ and $q$, both being 3 mod 4
2. Set $n = pq$
3. Choose a random integer $x$ coprime to $n$
4. Define
$$\begin{cases} x_0 & \equiv & x^2 \bmod n \\ x_{i+1} & \equiv & x_i^2 \bmod n \end{cases}$$
5. At each iteration select the least significant bit of $x_i$

## Why is BBS secure?

*Can bits generated using BBS be predicted?*

**Problem** (Quadratic Residuosity (QR))

Let $n = pq$ be the product of two primes. Given an integer $y$, is it a square mod $n$, i.e. is there an $x$ such that $x^2 \equiv y \bmod n$?

This loose formulation will be refined in the next chapter (3.166).

Strategy:

- Prove that the QR problem is hard
- If this is hard the previous bit cannot be predicted
- A sequence a pseudo-random bits generated by BBS cannot be compressed

## Reminder

In order to prove that the QR problem is hard we first recall and prove few results from number theory. The goal is to prove that solving the QR problem is as hard as factoring. That is, knowing how to solve one implies knowing how to solve the other one.

**Theorem** (Fermat's little theorem)

Let $p \in \mathbb{N}$ and $a \in \mathbb{Z}$. If $p$ is prime and $p \nmid a$, then

$$a^{p-1} \equiv 1 \bmod p.$$

More generally, for any prime $p \in \mathbb{N}$ and $a \in \mathbb{Z}$,

$$a^p \equiv a \bmod p.$$

## Squares modulo a prime

**Lemma**

If $p \equiv 3 \bmod 4$ is prime, then the equation $x^2 \equiv -1 \bmod p$ has no solution.

Proof. Suppose such an $x$ exists. Then raising it to the power of $(p-1)/2$ and applying Fermat's little theorem (2.95) yields

$$(x^2)^{\frac{p-1}{2}} \equiv x^{p-1} \equiv 1 \bmod p.$$

On the other hand $p \equiv 3 \bmod 4$, implies $(p-1)/2$ odd and

$$(-1)^{\frac{p-1}{2}} \equiv -1 \bmod p. ↯$$

□

## Square roots modulo a prime

**Proposition**

Let $p \equiv 3 \bmod 4$ be a prime, $y$ be an integer and $x \equiv y^{\frac{p+1}{4}} \bmod p$.

- If $y$ has a square root mod $p$, then its square roots are $\pm x \bmod p$
- If $y$ has no square root mod $p$, then the square roots of $-y$ are $\pm x \bmod p$

Proof. The case $y \equiv 0 \bmod p$ being trivial, we assume $y \not\equiv 0 \bmod p$. Applying Fermat's little theorem (2.95) we get

$$x^4 \equiv y^{p+1} \equiv y^2 y^{p-1} \equiv y^2 \bmod p. \tag{2.1}$$

## Square roots modulo a prime

Proof (continued). Since $p$ is prime all the non zero elements have a multiplicative inverse (prop. 1.52). Therefore rewriting eq. (2.1) into

$$(x^2 - y)(x^2 + y) \equiv 0 \bmod p,$$

implies $x^2 \equiv \pm y \bmod p$. Hence at least one of $y$ and $-y$ is a square mod $p$.

Suppose that both $y$ and $-y$ are square mod $p$, i.e. there exist $a$ and $b$ such that $y \equiv a^2 \bmod p$ and $-y \equiv b^2 \bmod p$.

Then $\left(b^{-1}a\right)^2 \equiv -1 \bmod p$, that is $-1$ is a square mod $p$, contradicting lem. 2.96$\lightning$.

Hence exactly one of $y$ and $-y$ has square roots $\pm x \bmod p$.

$\square$

## Reminder

Keeping in mind the initial goal of studying the BBS generator where the squares are computed mod $n = pq$, with both $p$ and $q$ congruent to 3 modulo 4, we recall the following result.

**Theorem** (Chinese Remainder Theorem (CRT))

Let $m_1, \dots, m_k \in \mathbb{N}\backslash\{0\}$ be pairwise relatively prime and $a_1, \dots, a_k \in \mathbb{Z}$. Then the system of congruences

$$\left\{\begin{array}{lcl} x & \equiv & a_1 \bmod m_1, \\ x & \equiv & a_2 \bmod m_2, \\ & \vdots & \\ x & \equiv & a_n \bmod m_k. \end{array}\right.$$

has a unique solution modulo $m = m_1 m_2 \dots m_k$.

## Square roots for a composite modulus

Example. Find $x$ such that $x^2 \equiv 71 \bmod 77$.

As $77 = 7 \times 11$, the congruency can be rewritten

$$\begin{cases} x^2 \equiv 71 \equiv 1 \bmod 7 \\ x^2 \equiv 71 \equiv 5 \bmod 11. \end{cases}$$

As both 7 and 11 and 3 mod 4, from prop. 2.97 we derive

$$\begin{cases} x \equiv \pm 1 \bmod 7 \\ x \equiv \pm 4 \bmod 11. \end{cases}$$

Finally, by applying the CRT (2.99) the four solutions can be recombined modulo 77 such as to get

$$x \equiv \pm 15,\ \pm 29 \bmod 77.$$

## From square roots to factors

In the previous example we used the factorisation of $n$ in order to calculate the square root of $x$ modulo $n$. We now show that if we know the square root then we can factorize $n$.

**Proposition**

Let $n$ be a product of two unknown primes $p$ and $q$, both being 3 mod 4. Let $x \equiv \pm a, \ \pm b \bmod n$ be the four solutions to $x^2 \equiv y \bmod n$. Then $\gcd(a-b, n)$ is a non-trivial factor of $n$.

Proof. From the construction of $a$ and $b$, we know that $a \equiv b \bmod p$ and $a \equiv -b \bmod q$ (or the other way around). Therefore $p|(a-b)$ while $q \nmid (a-b)$, which means that $\gcd(a-b, n) = p$. $\square$

## Remarks on the BBS generator

We showed that:

- Solving the factorization problem allows to solve the QR problem
- Solving the QR problem gives the factorization of the modulus

The previous reasoning is:

- Not a formal security reduction
- Enough to “informally” consider BBS as a secure pseudo-random number generator

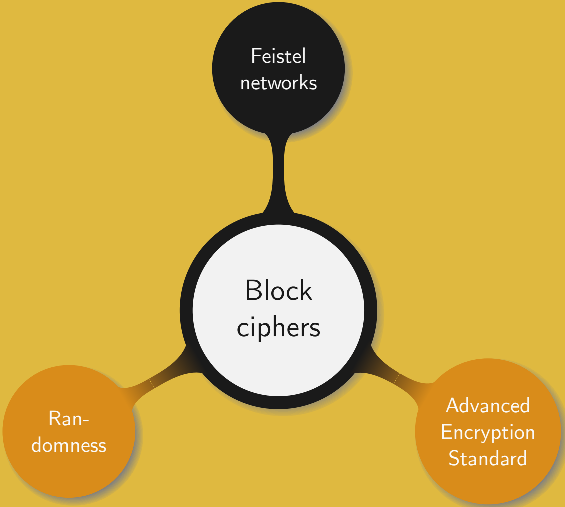
## Building a block cipher

A few informal definitions:

- A *random oracle* is a "black box" that returns a truly uniform random output on an input. Submitting the same input more than once leads to the same output.
- A *pseudorandom function* is a function that emulates a random oracle

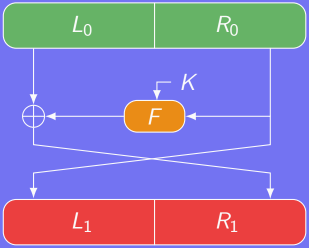
- A pseudorandom function that cannot be distinguished from a random permutation is called *pseudo random permutation*
- A *blockcipher* is a pseudorandom permutation
- A *one way function* is a function easy to evaluate but hard to invert

## Chapter organisation



## Basic Feistel network

*We want to build a random bijection over 2n bits*



- Size of a block: $2n$ bits
- Split the block into two blocks of $n$ bits each
- $F : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$

We define the function

$$\begin{aligned} \Psi_F : \{0,1\}^{2n} &\longrightarrow \{0,1\}^{2n} \\ [L, R] &\longmapsto [R, L \oplus F(R, K)] \end{aligned}$$

## Inverse function

**Proposition**

For any function $F$, $\Psi_F$ is a bijection and $\Psi_F^{-1} = \sigma \circ \Psi_F \circ \sigma$, with

$$\begin{aligned} \sigma : \{0,1\}^{2n} &\longrightarrow \{0,1\}^{2n} \\ [L, R] &\longmapsto [R, L] \end{aligned}$$

Proof. By definition of $\Psi_F$, $\Psi_F\left([L_0, R_0]\right) = [R_0, L_0 \oplus F(R_0, K)] = [L_1, R_1]$.
Equivalently,

$$\begin{cases} R_0 &= L_1 \\ L_0 &= R_1 \oplus F(L_1, K). \end{cases}$$

Moreover

$$\begin{aligned} \sigma \circ \Psi_F \circ \sigma([L_1, R_1]) &= \sigma \circ \Psi_F \circ \sigma([R_0, L_0 \oplus F(R_0, K)]) \\ &= \sigma\left(\Psi_F([L_0 \oplus F(R_0, K), R_0])\right) \\ &= \sigma\left(R_0, L_0 \oplus F(R_0, K) \oplus F(R_0, K)\right) \\ &= [L_0, R_0]. \end{aligned}$$

□

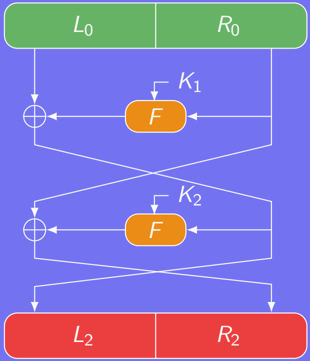
## Attacking Feistel networks

Setting up two black boxes, a random oracle and a Feistel network, the goal for an attacher is to distinguish them.

Number of messages necessary to reach the goal:

| Rounds | KPA | CPA | CPCA |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | $\mathcal{O}\left(\sqrt{2^n}\right)$ | 2 | 2 |
| 3 | $\mathcal{O}\left(\sqrt{2^n}\right)$ | $\mathcal{O}\left(\sqrt{2^n}\right)$ | 3 |
| 4 | $\mathcal{O}\left(2^n\right)$ | $\mathcal{O}\left(\sqrt{2^n}\right)$ | $\mathcal{O}\left(\sqrt{2^n}\right)$ |

## Two rounds – CPA



Attack strategy:

- For simplicity we denote $F(X,K)$ by $F_k(X)$ and $\Psi_{F_{k_2}} \circ \Psi_{F_{K_1}}$ by $\Psi^2_{F_{K_1},F_{K_2}}$
- $\Psi^2_{F_{K_1},F_{K_2}}\left([L_0,R_0]\right) = [L_2,R_2]$ with $L_2 = L_0 \oplus F_{K_1}(R_0)$ and $R_2 = R_0 \oplus F_{K_2}(L_2)$
- The inverse of $\Psi^2_{F_{K_1},F_{K_2}}$ is

$$\begin{aligned}\Psi^{-2}_{F_{K_1},F_{K_2}} &= \Psi^{-1}_{F_{K_1}} \circ \Psi^{-1}_{F_{K_2}}\\ &= \sigma \circ \Psi_{F_{K_1}} \circ \sigma \circ \sigma \circ \Psi_{F_{K_2}} \circ \sigma\\ &= \sigma \circ \Psi^2_{F_{K_2},F_{K_1}} \circ \sigma\end{aligned}$$

What if we use $m_1 = [m_{1_L}, m_{1_R}]$ and $m_2 = [m_{2_L}, m_{2_R}]$ such that

$$\begin{cases} m_{1_L} \neq m_{2_L}\\ m_{1_R} = m_{2_R}\end{cases}$$

## Two rounds – KPA

Basic KPA strategy:

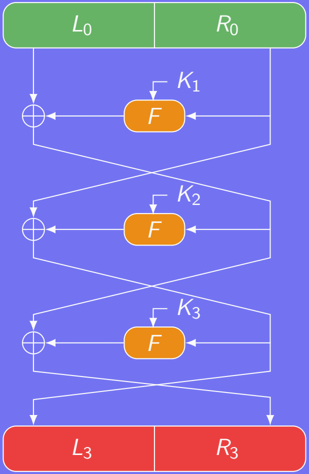
1. Find a collision over the $m_{i_R}$, $1 \leq i \leq 2^n$
2. If a collision is found for $m_j$ and $m_l$ check if

$$m_{j_{L_2}} \oplus m_{l_{L_2}} = m_{j_{L_0}} \oplus m_{l_{L_0}}$$

By the birthday paradox (slide 4.231):

- Number of plaintext-ciphertext pairs needed: $\mathcal{O}(\sqrt{2^n})$
- No better than $\mathcal{O}(\sqrt{2^n})$:
  - Collision on $m_{i_{L_2}} = m_{i_{L_0}} \oplus F_{K_1}(m_{i R_0})$ for two messages
  - The variables $m_{i_{L_2}}$, $m_{i_{L_0}}$, and $m_{i_{R_0}}$ are fixed
  - It only depends on $F_{K_1}$, which can take $2^n$ different values
  - From $l$ messages $\frac{l(l-1)}{2}$ pairs can be constructed
  - Probability of collision: $\approx \frac{l(l-1)}{2 \cdot 2^n}$

## Three rounds – CPCA



Attack strategy:

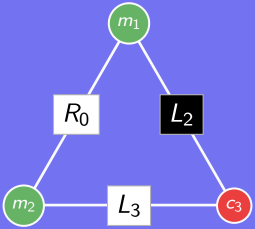
- $\Psi^3_{F_{k_1},F_{k_2},F_{k_3}}\left([L_0, R_0]\right) = [L_3, R_3]$ with
$$\begin{cases} L_3 = R_0 \oplus F_{K_2}(L_2) \\ R_3 = L_2 \oplus F_{K_3}(L_3) \end{cases}$$
and $L_2 = L_0 \oplus F_{K_1}(R_0)$

- Notice for a pair of messages $(m_a, m_b)$
$$\begin{aligned} m_{a_{R_0}} = m_{b_{R_0}} &\Leftrightarrow m_{a_{L_2}} \oplus m_{b_{L_2}} = m_{a_{L_0}} \oplus m_{b_{L_0}} \\ m_{a_{L_2}} = m_{b_{L_2}} &\Leftrightarrow m_{a_{L_3}} \oplus m_{b_{L_3}} = m_{a_{R_0}} \oplus m_{b_{R_0}} \qquad (2.2 \\ m_{a_{L_3}} = m_{b_{L_3}} &\Leftrightarrow m_{a_{R_3}} \oplus m_{b_{R_3}} = m_{a_{L_2}} \oplus m_{b_{L_2}} \end{aligned}$$

## Three rounds – CPCA

Attack strategy:

- Choose two plaintexts $m_1$ and $m_2$ such that $m_{2_{R_0}} = m_{1_{R_0}}$
- Choose a ciphertext $c_3$ such that $m_{3_{L_3}} = m_{2_{L_3}}$
- From the third equation of (2.2)
  $$m_{3_{R_3}} \oplus m_{2_{R_3}} = m_{3_{L_2}} \oplus m_{2_{L_2}}$$
- Using the first equation of (2.2) enforce $m_{3_{L_2}}$ to be $m_{1_{L_2}}$
  $$m_{3_{R_3}} = m_{2_{R_3}} \oplus m_{1_{L_0}} \oplus m_{2_{L_0}}$$
- How to conclude?

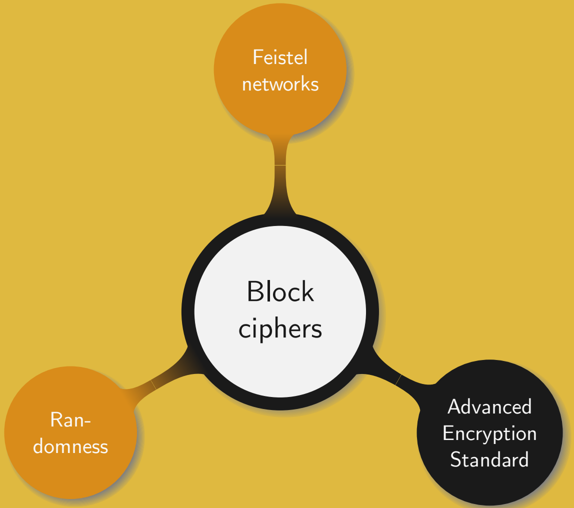


## Data Encryption Standard

Data Encryption Standard (DES):

- 1974: IBM uses Feistel networks to create LUCIFER
- 1975: LUCIFER is sent to NSA for review and modifications
- 1977: renamed DES and becomes the official encryption standard
- 2002: DES is not secure anymore and is replaced by AES

## Chapter organisation



## Advanced Encryption Standard

Advanced Encryption Standard (AES):

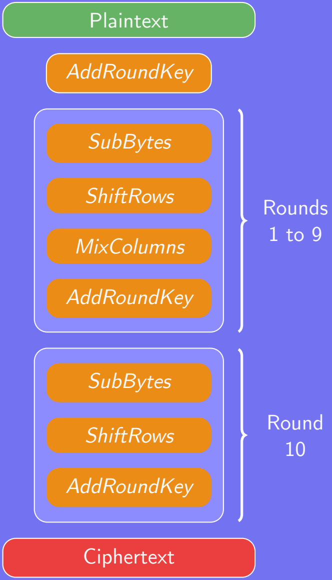
- 1997: call for candidates to replace DES
- Requirements:
  - Possible key sizes: 128, 192 and 256 bits
  - Input block size: 128 bits
  - Work on various hardware (e.g. 8-bit processors)
  - Speed
- Five finalists: MARS, RC6, Rijndael, Serpent, and Twofish
- 2001: Rijndael is chosen to become AES

## AES – General description

Brief outline of AES:

- 10 rounds for a 128-bit key (12 and 14 for 192 and 256-bit)
- A round is formed of layers
  - *SubBytes*: substitution operation
  - *ShiftRows*: linear mixing step on the rows
  - *MixColumns*: linear mixing on the columns
  - *AddRoundKey*: apply a round key derived from the main key

## AES – Encryption



AES setup:

- The 128 bits are grouped into 16 bytes
- Each byte is composed of 8 bits:

$$a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{1,1}, \cdots, a_{3,3}$$

- Bits are arranged in a $4 \times 4$ matrix:

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

## Invertible elements

So far we worked with the set $S = \{0, \cdots, n-1\}$ using modular congruences (def. 1.38). In the proof of prop. 2.97 we noted that when $n$ is prime all the non-zero elements of $S$ are invertible.

Example.

i The set $S = \{0, \cdots, 4\}$ has five elements, and since five is prime all the non-zero elements are invertible. Indeed,

$$1 \cdot 1 \equiv 1 \bmod 5, \;\; 2 \cdot 3 = 6 \equiv 1 \bmod 5, \text{ and } 4^2 = 16 \equiv 1 \bmod 5.$$

ii The set $S = \{0, \cdots, 5\}$ has six elements, and as six is not prime some non-zero elements are not invertible. In fact since

$$2 \cdot 3 = 6 \equiv 0 \bmod 6,$$

we conclude that 2 and 3 are not invertible mod 6.

## Finite fields

Loosely speaking a set where the addition and multiplication operations are defined and such that every non-zero element is invertible for the multiplication is called a *field*.

When a field has a finite number of elements it is called *finite field*. For each prime $p$ and positive integer $n$ there exists a finite field with $p^n$ elements, often denoted $\mathrm{GF}(p^n)$ or $\mathbb{F}_{p^n}$ (GF standing for Galois Field).
Remark. The set $S = \{0, \cdots, 8\}$ has $9 = 3^2$ elements and is not a field since 3 is not invertible. Therefore the question remaining to answer is "how to construct a finite field with nine elements", or more generally with $p^n$ elements.

## Polynomials over finite fields

Similarly to how polynomials are defined over common fields such as the real numbers, they can also be defined over finite fields. The main difference relies on their coefficients which take their values in the base field.

In a field, a polynomial which cannot be written as the product of two polynomials of lower degree is said to be *irreducible*.
Example.

i. In $\mathbb{F}_2[X]$, $X^2 + 3X + 1$ and $X^2 + X + 1$ are equal.
ii. In $\mathbb{F}_5[X]$, $X^3 + X + 3 = (X + 4)(X^2 + X + 2)$ is not irreducible.
iii. In $\mathbb{F}_{17}[X]$, $X^3 + X + 3$ is irreducible.

## Non-prime fields

**Theorem**

Let $P(X)$ be an irreducible polynomial of degree $n$ in $\mathbb{F}_p[X]$, and $F$ be the set of all the polynomials of degree less than $n$. Then $F$ is a finite field with $p^n$ elements.

Proof. Assuming addition and multiplication are properly defined we need to prove that $F$ has $p^n$ elements and that all but 0 are invertible.

It is simple to see that $F$ has $p^n$ elements since each of the $n$ monomials (from degree 0 to $n-1$) can take $p$ different values (from 0 to $p-1$).

## Non-prime fields

Proof (continued). Let $A(X)$, $B(X)$ and $C(X)$ be three distinct non-zero polynomials such that

$$A(X)B(X) \equiv A(X)C(X) \bmod P(X).$$

This implies $A(X)\left(B(X) - C(X)\right) \equiv 0 \bmod P(X)$, which is not possible since $P(X)$ is irreducible.

Hence multiplying a polynomial $A(X)$ by all the non-zero elements of $F$ results in covering all the non-zero polynomials of $F$, meaning that there is a polynomial $B(X)$ such that

$$A(X)B(X) \equiv 1 \bmod P(X).$$

$\square$

## Finite fields in the AES

In Rijndael $\mathbb{F}_{2^8}$ is used:

- $P(X) = X^8 + X^4 + X^3 + X + 1$ is the irreducible over $\mathbb{F}_2[X]$
- Each element of $\mathbb{F}_{2^8}$ is a polynomial of the form
$$a_7X^7 + a_6X^6 + a_5X^5 + a_4X^4 + a_3X^3 + a_2X^2 + a_1X + a_0$$
- The polynomial is described as a byte $a_7a_6a_5a_4a_3a_2a_1a_0$
- The sum of two polynomials is the XOR of their bit representation
- Multiplying a polynomial $Q(X)$ by $X$:
  1. Shift left the byte representation of $Q(X)$ and append a 0
  2. If the first bit is 0 stop and otherwise XOR with $P(X)$
- Multiplying $Q(X)$ by $R(X)$:
  1. Split $R(X)$ into the monomials $M_i(X)$, $i \leq \deg R(X)$
  2. For $M_i(X)$ applying the multiplication by $X$ $\deg M_i(X)$ times
  3. Add all the results using XOR

## Finite fields in the AES

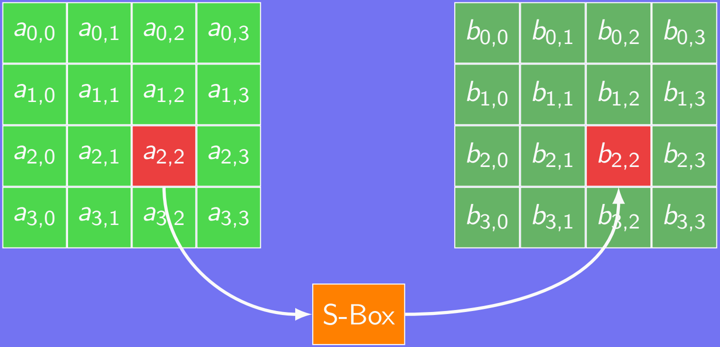
Example. Let $Q(X) = X^7 + X^4 + X + 1$ and $R(X) = X^2 + 1$. Determine the product $Q(X)R(X)$ in $\mathbb{F}_{2^8}[X]$.

1. Regular strategy: multiply and reduce mod $P(X)$
   - $Q(X)R(X) = X^9 + X^7 + X^6 + X^4 + X^3 + X^2 + X + 1$
   - Since $P(X) = 0$, $X^9 = X^5 + X^4 + X^2 + X$ and
   $$Q(X)R(X) \equiv X^7 + X^6 + X^5 + X^3 + 1 \bmod P(X)$$
2. Represent polynomials as bytes and apply XOR operations:

Write $Q(X) = 10010011$ and decompose $R(X)$ as $X \cdot X + 1$

- $Q(X) \cdot X = 100100110 \oplus 100011011 = 000111101$
- $(Q(X) \cdot X) \cdot X = 001111010$
- $(Q(X) \cdot X) \cdot X + Q(X) = 01111010 \oplus 10010011 = 11101001$
- $Q(X)R(X) \equiv X^7 + X^6 + X^5 + X^3 + 1 \bmod P(X)$

## The SubBytes layer



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
S-Box
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

For each byte in the matrix:

- Split it into two 4-bit numbers $a$ and $b$
- Find byte $c$ in the S-Box table at row $a$ and column $b$
- Replace the original byte by $c$

## S-Box

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 99 | 124 | 119 | 123 | 242 | 107 | 111 | 197 | 48 | 1 | 103 | 43 | 254 | 215 | 171 | 118 |
| 1 | 202 | 130 | 201 | 125 | 250 | 89 | 71 | 240 | 173 | 212 | 162 | 175 | 156 | 164 | 114 | 192 |
| 2 | 183 | 253 | 147 | 38 | 54 | 63 | 247 | 204 | 52 | 165 | 229 | 241 | 113 | 216 | 49 | 21 |
| 3 | 4 | 199 | 35 | 195 | 24 | 150 | 5 | 154 | 7 | 18 | 128 | 226 | 235 | 39 | 178 | 117 |
| 4 | 9 | 131 | 44 | 26 | 27 | 110 | 90 | 160 | 82 | 59 | 214 | 179 | 41 | 227 | 47 | 132 |
| 5 | 83 | 209 | 0 | 237 | 32 | 252 | 177 | 91 | 106 | 203 | 190 | 57 | 74 | 76 | 88 | 207 |
| 6 | 208 | 239 | 170 | 251 | 67 | 77 | 51 | 133 | 69 | 249 | 2 | 127 | 80 | 60 | 159 | 168 |
| 7 | 81 | 163 | 64 | 143 | 146 | 157 | 56 | 245 | 188 | 182 | 218 | 33 | 16 | 255 | 243 | 210 |
| 8 | 205 | 12 | 19 | 236 | 95 | 151 | 68 | 23 | 196 | 167 | 126 | 61 | 100 | 93 | 25 | 115 |
| 9 | 96 | 129 | 79 | 220 | 34 | 42 | 144 | 136 | 70 | 238 | 184 | 20 | 222 | 94 | 11 | 219 |
| 10 | 224 | 50 | 58 | 10 | 73 | 6 | 36 | 92 | 194 | 211 | 172 | 98 | 145 | 149 | 228 | 121 |
| 11 | 231 | 200 | 55 | 109 | 141 | 213 | 78 | 169 | 108 | 86 | 244 | 234 | 101 | 122 | 174 | 8 |
| 12 | 186 | 120 | 37 | 46 | 28 | 166 | 180 | 198 | 232 | 221 | 116 | 31 | 75 | 189 | 139 | 138 |
| 13 | 112 | 62 | 181 | 102 | 72 | 3 | 246 | 14 | 97 | 53 | 87 | 185 | 134 | 193 | 29 | 158 |
| 14 | 225 | 248 | 152 | 17 | 105 | 217 | 142 | 148 | 155 | 30 | 135 | 233 | 206 | 85 | 40 | 223 |
| 15 | 140 | 161 | 137 | 13 | 191 | 230 | 66 | 104 | 65 | 153 | 45 | 15 | 176 | 84 | 187 | 22 |

## Generating the S-Box

Simple construction:

- For $a$ in $\mathbb{F}_{2^8}^*$ compute its inverse $b = a^{-1}$ or set $b = 0$ if $a = 0$
- Represent $b$ as a column vector $B = (b_0, \cdots, b_7)$
- Compute

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{pmatrix}$$

- The entry located at row $(a_7 \cdots a_4)_2$ and column $(a_3 \cdots a_0)_2$ of the S-Box is $(c_7 \cdots c_0)_2$

## Generating the S-Box

Example. Find the S-Box entry corresponding to the byte 11001011?

The byte 11001011 stands for $a(X) = X^7 + X^6 + X^3 + X + 1$, and we observe that

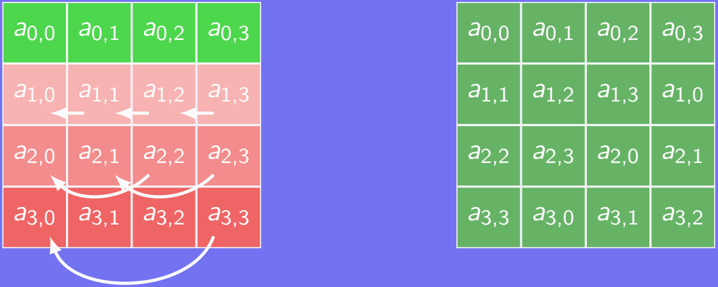
$$\begin{aligned} a(X) \cdot X^2 &= X^9 + X^8 + X^5 + X^3 + X^2 \\ &\equiv X^8 + X^4 + X^3 + X \bmod P(X) \\ &\equiv 1 \bmod P(X). \end{aligned}$$

Therefore we calculate

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

and finally conclude that the entry at row 12 and column 11 is 31.

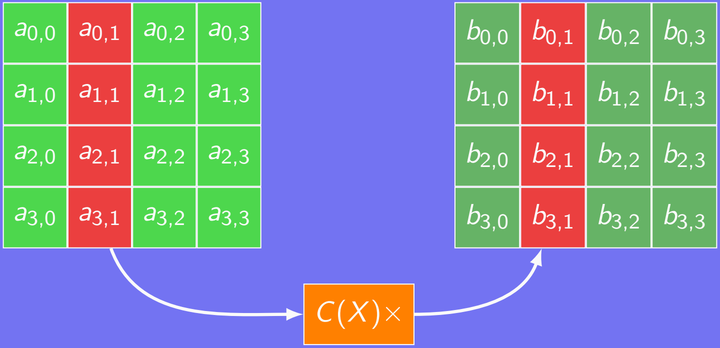
## The ShiftRows layer



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Cyclically shift to the left row $i$ by offset $i$, $0 \leq i \leq 3$

## The MixColumns layer

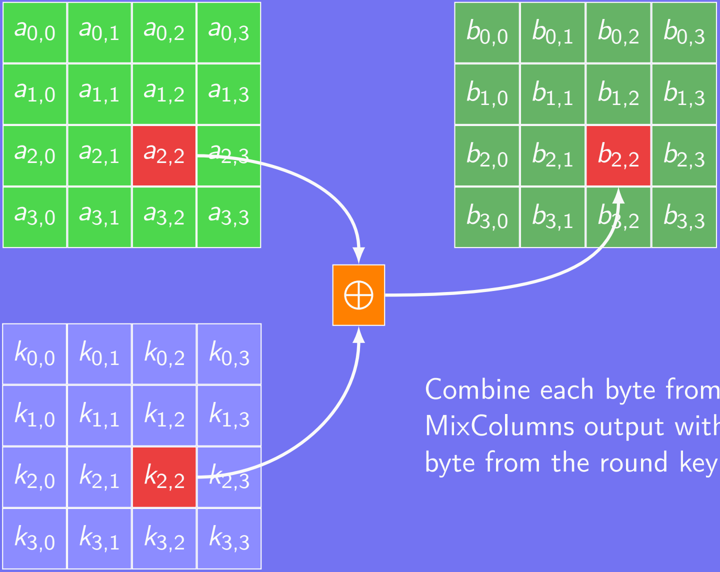


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$C(X)\times$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Left multiply the output of ShiftRows by the matrix

$$C(X) = \begin{pmatrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{pmatrix}$$

## The AddRoundKey layer



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$\oplus$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Combine each byte from the MixColumns output with a byte from the round key

## Generating a round key

The original 128 bits key is arranged into a $4 \times 4$ matrix $K(X)$

Label the first four columns $K(0), \cdots, K(3)$ and add forty more:

- $K(i) = K(i-4) \oplus K(i-1)$, for $i \not\equiv 0 \bmod 4$
- $K(i) = K(i-4) \oplus T(K(i-1))$, for $i \equiv 0 \bmod 4$

The transformation $T(K(i-1))$ is defined over the column $i$:

- Compute $r(i) = 00000010^{\frac{i-4}{4}}$
- Cyclically top shift the elements of the column by 1
- Apply the SubBytes layer (2.124) to each byte of the column and get the column vector $(a, b, c, d)$
- Finally return the column vector

$$T(K(i-1)) = (a \oplus r(i), b, c, d)$$

The $i$-th round key is given by the columns $K(4i), \cdots, K(4i+3)$

## Generating a round key

Example. $K(i)$ being simple to generate for $i \not\equiv 0 \bmod 4$, we focus on the case $i \equiv 0 \bmod 4$. For instance if $i = 40$ and $K(39)$ is the column vector $(10001100, 00001100, 11000110, 11110011)$, then

- Cyclical top shit: $(00001100, 11000110, 11110011, 10001100)$
- SubBytes transformation:
$$\begin{array}{ccccccc} 00001100 & \rightarrow & 11111110, & & 11000110 & \rightarrow & 10110100 \\ 11110011 & \rightarrow & 00001101, & & 10001100 & \rightarrow & 01100100 \end{array}$$
- $r(40) = X^9 \equiv X^5 + X^4 + X^2 + X \bmod P(X) = 00110110$
- Get the final column vector $T(K(39))$
$$\begin{aligned} T(K(39)) &= (11111110 \oplus 00110110, 10110100, 00001101, 01100100) \\ &= (11001000, 10110100, 00001101, 01100100) \end{aligned}$$
- Finally define $K(40)$ as $K(36) \oplus T(K(39))$

## Decryption

The decryption process is simple:

- Perform all the operations in reverse order
- Replace the SubBytes, ShitRows and MixColumns operations by their inverse

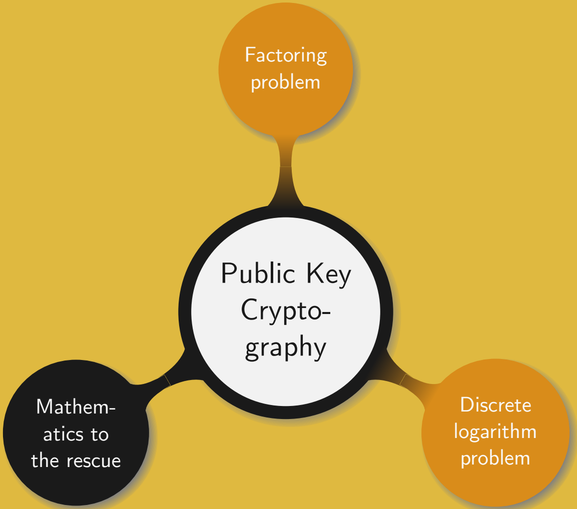
Remark. It is possible to construct an inverse cipher performing decryption by applying a sequence of inverse operations in the same order as it is done for encryption

## Key points

- What does it mean to be random?
- Recall Fermat's little theorem
- Recall the CRT
- How is a Feistel network organised?
- Describe AES

## 3. Public Key Cryptography

## Chapter organisation



## Reminder

Symmetric-key cryptosystem:

- Encryption depends on a secret key $K$
- Decryption depends on a secret key $K'$ easily derived from $K$
- Knowing $K$ or $K'$ is the same
- The key must be securely shared beforehand

Public-key cryptosystem:

- Encryption depends on a public key $K$
- Decryption depends on a secret key $K'$
- Finding $K'$ when knowing $K$ is computationally infeasible
- No prior communication is required

## One way functions

*One way function:* function easy to evaluate but hard to invert

Requirements for encrypting with a one-way function $E$:

- $E$ must be injective
- Some secret that allows to invert $E$

*Trapdoor one-way function:* one-way function, easy to invert when knowing a *trapdoor*

**Definition** (Group)

A *group* is a pair $(G, \circ)$ consisting of a set $G$ and a *group operation* $\circ\colon G \times G \to G$ that verifies the following properties:

i. *Associativity:* $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a, b, c \in G$
ii. *Existence of a unit element:* there exists an element $e \in G$ such that $a \circ e = e \circ a = a$ for all $a \in G$
iii. *Existence of inverse:* for every $a \in G$ there exists an element $a^{-1} \in G$ such that $a \circ a^{-1} = a^{-1} \circ a = e$

A group is called *abelian* if in addition to the above properties

iv. *Commutativity:* $a \circ b = b \circ a$ for all $a, b \in G$.

## Mathematical structures

**Definition** (Ring)

A *ring* is a triple $(R, +, \cdot)$ consisting of a set $R$ and two *binary operations* $+, \cdot : R \times R \to R$ such that

- i $(R, +)$ is an abelian group
- ii *Multiplicative unit:* there exists an element $1 \in R$ such that
$$a \cdot 1 = 1 \cdot a = a \qquad \text{for all } a \in R$$
- iii *Associativity:* for any $a, b, c \in R$,
$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$
- iv *Distributivity:* for any $a, b, c \in R$,
$$a \cdot (b + c) = (a \cdot b) + (a \cdot c), \quad (b + c) \cdot a = (b \cdot a) + (c \cdot a)$$

A ring is called *commutative* if in addition to the above properties

- v *Commutativity:* $a \cdot b = b \cdot a$ for all $a, b \in R$

**Definition** (Field)

Let $(F, +, \cdot)$ be a commutative ring with unit element of addition 0 and unit element of multiplication 1. Then $F$ is a *field* if

i. $0 \neq 1$
ii. For every $a \in F \setminus \{0\}$ there exists an element $a^{-1}$ such that
$$a \cdot a^{-1} = 1.$$

Remark. Another way of writing this definition is to say that $(F, +, \cdot)$ is a field if $(F, +)$ and $(F \setminus \{0\}, \cdot)$ are abelian groups and $0 \neq 1$, and $\cdot$ distributes over $+$.

## Mathematical structures

Example. Let $n$ be an integer, and $\mathbb{Z}/n\mathbb{Z}$ be the set of the integers modulo $n$

- $(\mathbb{Z}/n\mathbb{Z}, +)$ also denoted $(\mathbb{Z}_n, +)$ is a group
- $(\mathbb{Z}/n\mathbb{Z}, +, \cdot)$ is a ring
- If $n$ is prime then $(\mathbb{Z}/n\mathbb{Z}, +, \cdot)$ is the field $\mathbb{F}_n$
- The invertible elements of $\mathbb{Z}/n\mathbb{Z}$, with respect to '$\cdot$', form a group denoted $\mathrm{U}(\mathbb{Z}/n\mathbb{Z})$ or sometimes $\mathbb{Z}_n^\times$ or $\mathbb{Z}_n^*$
- $(\mathbb{Z}/n\mathbb{Z}[X], +, \cdot)$ is the ring of the polynomials over $\mathbb{Z}/n\mathbb{Z}$
- If $n$ is prime and the polynomial $P(X)$ is irreducible over $\mathbb{F}_n[X]$, then $(\mathbb{F}_n[X]/\langle P(X)\rangle, +, \cdot)$ is a field; this is $\mathbb{F}_{n^{\deg P(x)}}$

## Order

**Definitions**

Let $G$ be a group.

1. The *order* of $G$ is its cardinality
2. The *order* of an element $g \in G$ is the smallest positive integer $m$ such that $g^m = 1$
3. An element of order equal to the order of the group is called a *primitive element* or a *generator*
4. When $G = \mathbb{Z}/n\mathbb{Z}$, *Euler's totient function* $\varphi(n)$ counts the number of invertible elements, that is the number of elements $k$ such that $\gcd(n, k) = 1$

## Order

Example. The order of $U(\mathbb{Z}/13\mathbb{Z}) = 12$ and 2 is a generator:

| $i$ | $2^i \bmod 13$ | $i$ | $2^i \bmod 13$ | $i$ | $2^i \bmod 13$ | $i$ | $2^i \bmod 13$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 3 | 7 | 11 | 10 | 10 |
| 2 | 4 | 5 | 6 | 8 | 9 | 11 | 7 |
| 3 | 8 | 6 | 12 | 9 | 5 | 12 | 1 |

Remark. Let $p$ be a prime and $\alpha$ be a generator of $G = U(\mathbb{Z}/p\mathbb{Z})$. Then any element $\beta \in G$ can be written $\beta = \alpha^i$, $1 \le i \le p-1$. Noting $d = \gcd(i, p-1)$ we have

$$\beta^{\frac{p-1}{d}} = \left(\alpha^i\right)^{\frac{p-1}{d}} = \left(\alpha^{p-1}\right)^{\frac{i}{d}} = 1.$$

Suppose that the order of $\beta$ divides $\frac{p-1}{d}$. Then $\operatorname{ord}(\beta) = \frac{p-1}{kd}$ for some $k > 1$ such that $kd \nmid i$, meaning that $\frac{p-1}{k} \cdot \frac{i}{d}$ is not a multiple of $p-1$. ↯
Hence the order of $\beta$ is $\frac{p-1}{d}$.

## Revisiting the CRT

In theorem 2.99 we recalled that a system of congruences has a unique solution modulo the product of all the moduli of the system. In fact this result can be rephrased in term of group structure.

We first recall that an *isomorphism* is a bijection that preserves algebraic structures.

**Theorem** (Chinese Remainder theorem (CRT))

Let $n$ be a positive integer with prime decomposition $n = \prod_i p_i^{e_i}$. Then there exists a ring *isomorphism* between $\mathbb{Z}/n\mathbb{Z}$ and $\prod_i \mathbb{Z}/p_i^{e_i}\mathbb{Z}$.

## Order of the group of units

From the previous theorem (3.145)

$$\mathrm{U}(\mathbb{Z}/n\mathbb{Z}) \approx \mathrm{U}\left(\prod_i \mathbb{Z}/p_i^{e_i}\mathbb{Z}\right).$$

Noting that a non invertible element of $\mathbb{Z}/p_i^{e_i}\mathbb{Z}$ is of the form $kp_i$ for some integer $k$, it cannot be coprime to $n$ and as such is not invertible modulo $n$. Conversely an element that is not invertible mod $n$ is a multiple of some $p_i$. Therefore

$$\mathrm{U}(\mathbb{Z}/n\mathbb{Z}) \approx \mathrm{U}\left(\prod_i \mathbb{Z}/p_i^{e_i}\mathbb{Z}\right) \approx \prod_i \mathrm{U}\left(\mathbb{Z}/p_i^{e_i}\mathbb{Z}\right).$$

**Proposition**

If $m$ and $n$ are two coprime integers then $\varphi(mn) = \varphi(m)\varphi(n)$. In particular if $m$ and $n$ are prime $\varphi(mn) = (m-1)(n-1)$.

## Lagrange's theorem

Having a way to determine the order of $\mathrm{U}(\mathbb{Z}/n\mathbb{Z})$, we now focus on the order of its elements. We first recall a fundamental result from group theory.

**Theorem** (Lagrange's theorem)

Let $G$ be a finite group and $H$ be a subgroup of $G$. Then the order of $H$ divides the order of $G$.

Noting that each element $x$ of $G$ generates a subgroup of order $\operatorname{ord}_G x$, it follows that the order of any element $x$ of $G$ divides the order of $G$.

Using Lagrange's theorem it is then possible to derive a result to quickly verify whether an invertible element modulo a prime $p$ is a generator of $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$. But first we provide an example and then extend Fermat's little theorem (2.95).

## Lagrange's theorem

Example. For $n = 5$, $\mathrm{U}(\mathbb{Z}/5\mathbb{Z}) = \{1, 2, 3, 4\}$ which is a group of order 4. Therefore each of those four elements generates a subgroup of $\mathrm{U}(\mathbb{Z}/5\mathbb{Z})$. Moreover these subgroups will have order 1, 2, or 4, since 4 is divisible by 1, 2, and 4.

In fact we have

$$\begin{aligned} \langle 1 \rangle &= \{1\}\,, & \langle 2 \rangle &= \{2, 4, 3, 1\}\,, \\ \langle 4 \rangle &= \{4, 1\}\,, & \langle 3 \rangle &= \{3, 4, 2, 1\}\,. \end{aligned}$$

That is, we have two groups of order 4 ($\langle 2 \rangle$ and $\langle 3 \rangle$), one group of order 2 ($\langle 4 \rangle$), and one group of order 1 ($\langle 1 \rangle$).

In particular note that the order of an element is equal to the order of the subgroup it generates.

## Euler's theorem

**Theorem** (Euler's theorem)

Let $a$ and $n$ be two coprime integers. Then
$$a^{\varphi(n)} \equiv 1 \bmod n.$$

Proof. From the previous reasoning on Lagrange's theorem (3.147) there exists $k > 0$ such that $k|\varphi(n)$ and $a^k = 1$. Writing $\varphi(n) = kl$ for some integer $l$ we have
$$a^{\varphi(n)} = a^{kl} \equiv \left(a^k\right)^l \equiv 1^l = 1 \bmod n.$$
$\square$

## Simple calculation

Example. Calculate $2^{639613} \bmod 5353$.

First we note that 5353 can be written as the product of two primes: 101 and 53. Therefore $\varphi(5353) = 100 \cdot 52 = 5200$.

Observing that $639613 \equiv 13 \bmod 5200$ we need to consider $2^{13} \bmod 5353$.

As $2^{13} = 8192$ we obtain $2^{639613} \equiv 2839 \bmod 5353$.

Remark. The previous discussion can be simply summarized as follows: when working modulo $n$, the exponent must be considered mod $\varphi(n)$.

## Finding primitive elements

**Theorem**

Let $p > 2$ be a prime and $\alpha \in \mathrm{U}(\mathbb{Z}/p\mathbb{Z})$. Then $\alpha$ is a generator of $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$ if and only if for all primes $q$ such that $q|(p-1)$, $\alpha^{(p-1)/q} \not\equiv 1 \bmod p$.

Proof.

($\Rightarrow$) Since $\alpha$ is a generator, for all $1 \le i < p-1$, $\alpha^i \not\equiv 1 \bmod p$.

($\Leftarrow$) Suppose that $\alpha$ is invertible but does not generate $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$. Calling its order $d$, the fraction $(p-1)/d$ defines an integer larger than 1. This is true because $d|(p-1)$ (Lagrange's theorem (3.147)) and $d < (p-1)$. If $q$ is a prime divisor of $(p-1)/d$, then $d$ divides $\frac{p-1}{q}$. So $\alpha^{(p-1)/q} \equiv 1 \bmod p$.

□

## Finding primitive elements

**Corollary**

Let $\alpha$ be a generator of $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$.

1. Let $n$ be an integer. Then $\alpha^n \equiv 1 \bmod p$ if and only if $n \equiv 0 \bmod (p-1)$.
2. Let $j$ and $k$ be two integers. Then $\alpha^j \equiv \alpha^k \bmod p$ if and only if $j \equiv k \bmod (p-1)$.

Proof. (1) This is straightforward from the previous theorem (3.151).

(2) Without loss of generality assume $j \geq k$. First suppose that $\alpha^j \equiv \alpha^k \bmod p$. Dividing both sides by $\alpha^k$ yields $\alpha^{j-k} \equiv 1 \bmod p$. From (1) we have $j - k \equiv 0 \bmod (p-1)$.

Conversely if $j \equiv k \bmod (p-1)$ then $j - k \equiv 0 \bmod (p-1)$, and by (1) $\alpha^{j-k} \equiv 1 \bmod p$. Finally we multiply by $\alpha^k$. $\square$

## Order and factorization

We now relate the order of the elements in $\mathrm{U}(\mathbb{Z}/n\mathbb{Z})$, where $n$ is a composite integer, to factoring $n$.

Let $x$ be an element of order $r$ in $\mathrm{U}(\mathbb{Z}/n\mathbb{Z})$. By definition we have $x^r \equiv 1 \bmod n$, that is $n|(x^r - 1)$.

If the order $r$ is even then $x^r - 1 = (x^{r/2} - 1)(x^{r/2} + 1)$. In this case both $\gcd(x^{r/2} - 1, n)$ and $\gcd(x^{r/2} + 1, n)$ are factors of $n$.

Conversely knowing the factorization of $n$ gives $\varphi(n)$. Since the order of an element $x$ in $\mathrm{U}(\mathbb{Z}/n\mathbb{Z})$ divides $\varphi(n)$ it suffices to write $\varphi(n) = \prod_i p_i$, where the $p_i$ are the prime factors of $\varphi(n)$. Then calculate $x^{a/p_i} \bmod n$, with $a = \varphi(n)$. If $x^{\varphi(n)/p_k} \equiv 1 \bmod n$, for some $k$, then redefine $a$ as $a/p_k$.

When all the $p_i$ have been tested $a$ defines the order of $x$. If none of the $x^{\varphi(n)/p_i} \bmod n$ is 1 then $x$ is a generator, i.e. has order $\varphi(n)$.

## Square roots modulo $p$

The previous discussion highlights the difficulty of determining the order of a random element of $\mathrm{U}(\mathbb{Z}/n\mathbb{Z})$, since it is equivalent to factoring $n$.

Another hard problem related to factorization was presented in chapter 2, namely the QR problem (2.94). In that chapter we studied the case where the primes are congruent to 3 modulo 4.

We now provide a more general result that gives a method to determine whether or not an element is a square modulo an arbitrary prime $p$.

**Proposition**

For $p$ an odd prime and $a$ such that $a \not\equiv 0 \bmod p$, $a^{\frac{p-1}{2}} \equiv \pm 1 \bmod p$. Moreover $a$ is a square mod $p$ if and only if $a^{\frac{p-1}{2}} \equiv 1 \bmod p$.

## Square roots modulo $p$

Proof. Defining $y \equiv a^{\frac{p-1}{2}} \bmod p$ and applying Fermat's little theorem (2.95), we have $y^2 \equiv a^{p-1} \equiv 1 \bmod p$. Therefore we have

$$y^2 - 1 \equiv (y-1)(y+1) \equiv 0 \bmod p.$$

As $p$ is prime all the elements but 0 are invertible, meaning that either $y \equiv 1 \bmod p$ or $y \equiv -1 \bmod p$.

If $a \equiv x^2 \bmod p$, then $a^{\frac{p-1}{2}} \equiv x^{p-1} \equiv 1 \bmod p$.

Conversely let $g$ be a generator mod $p$ and write $a \equiv g^j$ for some $j$. If $a^{\frac{p-1}{2}} \equiv 1 \bmod p$, then

$$g^{j\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} \equiv 1 \bmod p.$$

From corollary 3.152 we see that $j\frac{p-1}{2} \equiv 0 \bmod (p-1)$ implying that $j$ must be even. Hence $a \equiv g^j \equiv g^{2k}$ and $a$ is a square. $\square$

## Legendre symbol

Proposition 3.154 provides a simple way to computationally check if an element is a square modulo a prime. Since this criteria is difficult to use by hand we now introduce an alternative strategy.

**Definition** (Legendre symbol)

Given $p$ be an odd prime and $a \not\equiv 0 \bmod p$, we define the *Legendre symbol* by

$$\left(\frac{a}{p}\right) = \begin{cases} +1 \text{ if a is a square mod } p \\ -1 \text{ if a is not a square mod } p \end{cases}$$

## Legendre symbol

**Proposition**

Let $p$ be an odd prime.

1. If $a \equiv b \bmod p$, then $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.
2. If $a \not\equiv 0 \bmod p$, then $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \bmod p$.
3. If $ab \not\equiv 0 \bmod p$, then $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$.
4. If $p \equiv 1 \bmod 4$ then $-1$ is a square mod $p$.

Proof.

1. The solutions to the congruence $x^2 \equiv a \bmod p$ and $x^2 \equiv b \bmod p$ are the same when $a \equiv b \bmod p$.
2. Combining the definition of Legendre symbol (3.156) with proposition 3.154 yields the result.

## Legendre symbol

Proof (continued).

3. From (2), we have
$$\left(\frac{ab}{p}\right) = (ab)^{\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} b^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \bmod p.$$
Both ends being congruent to $\pm 1$ modulo the odd prime $p$ they are equal.

4. Applying (2) with $a = -1$ we have
$$\left(\frac{-1}{p}\right) \equiv (-1)^{\frac{p-1}{2}} \bmod p.$$
Again, since both ends are congruent to $\pm 1$ modulo the odd prime $p$ they are equal. Noting that when $p \equiv 1 \bmod 4$, $(p-1)/2$ is even gives the result. $\square$

## Legendre symbol

Example. Is 12 a square mod 31?

Since $12 = 2^2 \cdot 3$ we write

$$\left(\frac{12}{31}\right) = \left(\frac{2}{31}\right)^2 \left(\frac{3}{31}\right).$$

Moreover

$$\left(\frac{3}{31}\right) \equiv 3^{15} \equiv -1 \bmod 31.$$

Hence 12 is not a square mod 31.

## Extending Legendre symbol

In definition 3.156 the Legendre symbol is defined for primes. We would like to extend this definition to any odd integer $n$.

As a first attempt we define the symbol to be $+1$ if an integer $a$ is a square and $-1$ otherwise.

Example. Is 6 a square mod 35?

Noting that $6 = 2 \cdot 3$ we need to consider whether 2 and 3 are squares mod 35. In fact neither of them is, since they are not squares mod 5.

Similarly 6 is not a square mod 7, and as such cannot be a square mod 35.

Consequently, none of 2, 3, and 6 is a square mod 35, implying the third property of proposition 3.157 to give $(-1) \cdot (-1) = -1$. ↯

## Jacobi symbol

To preserve the third property of the Legendre symbol (3.157) we define the Jacobi symbol as follows.

**Definition** (Jacobi symbol)

Given $n = \prod_i p_i^{e_i}$ an odd integer and $a$ a non-zero integer coprime to $n$, we define the *Jacobi symbol* by

$$\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right)^{e_i},$$

where each of the $\left(\frac{a}{p_i}\right)$ is a Legendre symbol.

## Jacobi symbol

Remark.

- When $n$ is prime the Jacobi symbol reduces to the Legendre symbol
- Let $n = 135 = 3^3 \cdot 5$. Then
$$\left(\frac{2}{135}\right) = \left(\frac{2}{3}\right)^3 \left(\frac{2}{5}\right) = (-1)^3(-1) = 1.$$
However 2 is not a square mod 135 since it is not a square mod 5.
Hence a value of $+1$ for the Jacobi symbol does not imply that an integer is a square mod $n$.

## Jacobi symbol

**Proposition**

Let $n$ be an odd integer.

1. If $a \equiv b \bmod n$ and $\gcd(a, n) = 1$, then $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$.
2. If $\gcd(ab, n) = 1$ then $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$.
3. $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$.
4. $\left(\frac{2}{n}\right) = \begin{cases} +1 \text{ if } n \equiv 1 \text{ or } 7 \bmod 8 \\ -1 \text{ if } n \equiv 3 \text{ or } 5 \bmod 8 \end{cases}$
5. If $m$ and $n$ are odd coprime positive integers, then
$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right) & \text{if } m \equiv n \equiv 3 \bmod 4 \\ +\left(\frac{n}{m}\right) & \text{otherwise} \end{cases}$$

## Jacobi symbol

**Example.** Calculate $\left(\frac{4567}{12345}\right)$.

$$\begin{aligned}
\left(\frac{4567}{12345}\right) &= +\left(\frac{12345}{4567}\right) && \text{by (5), since } 12345 \equiv 1 \bmod 4\\
&= +\left(\frac{3211}{4567}\right) && \text{by (1), since } 12345 \equiv 3211 \bmod 4567\\
&= -\left(\frac{1356}{3211}\right) && \text{by (5) and (1)}\\
&= -\left(\frac{2}{3211}\right)^2\left(\frac{339}{3211}\right) && \text{by (2)}\\
&= +\left(\frac{3211}{339}\right) && \text{since } (\pm 1)^2 = 1 \text{ and by (5)}\\
&= +\left(\frac{2}{339}\right)^5\left(\frac{5}{339}\right) && \text{by (5) and since } 2^5 \cdot 5 = 160 \equiv 3211 \bmod 339\\
&= -\left(\frac{2}{5}\right)^2 && \text{by (4), (5), and (2)}\\
&= -1
\end{aligned}$$

## Jacobi symbol

Remark.

- In proposition 3.163 the fifth point is called the *quadratic reciprocity law*. When $m$ and $n$ are primes it relates the question of $m$ being a square mod $n$ to the one of $n$ being a square mod $m$.

- Let $n$ be the product of two primes $p$ and $q$ and $a$ be an integer. If $\left(\frac{a}{n}\right) = -1$, then $a$ is not a square mod $n$. What can be concluded if $\left(\frac{a}{n}\right) = +1$?

  As $\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$, either
  $$\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1 \text{ or } \left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = +1.$$
  In the first case $a$ is not a square mod $p$ and as such cannot be a square mod $n$, while in the second case $a$ is a square.

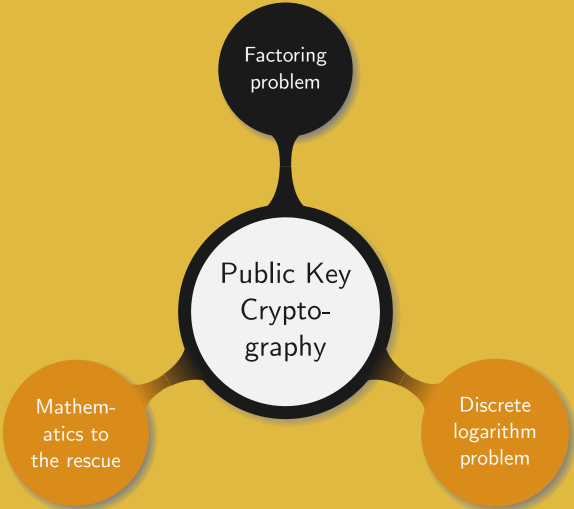
## The Quadratic Residuosity Problem

From the previous remark (3.165) we see that if $\left(\frac{a}{n}\right) = +1$ then $a$ can be either a square or a non-square. Deciding which one holds is known as the *Quadratic Residuosity Problem*, loosely introduced in problem 2.94.

**Problem** (Quadratic Residuosity (QR))

Let $n = pq$ be the product of two primes. Let $y$ be an integer such that $\left(\frac{y}{n}\right) = 1$. Determining whether or not $y$ is a square modulo $n$ is called the *Quadratic Residuosity Problem*.

## Chapter organisation



## From mathematics to cryptography

From the previous mathematical discussions in chapters 1 and 3 we know that given two primes $p$ and $q$, it is easy to compute their product $n$ as well as $\varphi(n)$ (proposition 3.146).

Then if an integer $e$, coprime to $\varphi(n)$, is chosen, it suffices to run the extended Euclidean algorithm (1.51) in order to determine the integer $d$ such that $ed \equiv 1 \bmod \varphi(n)$.

Therefore given $e$ and $n$ it is possible to compute $c \equiv m^e \bmod n$ for any integer $m$. Then computing $c^d \bmod n$ yields $m$ since

$$c^d \equiv (m^e)^d \equiv m^{ed \bmod \varphi(n)} \equiv m \bmod n.$$

The goal is now to use this mathematical setup in order to build a trapdoor one-way function and design a public key cryptosystem.

## Toward a new cryptosystem

Intuition:

- Generate $p$ and $q$, then compute $n$ and $\varphi(n)$
- Choose $e$ coprime to $\varphi(n)$ and determine $d$
- Anybody can encrypt: $n$ and $e$ are public
- Only one person can decrypt: $d$ is secret

Questions:

- How to effectively define the cryptosystem?
- Can the modular exponentiations to encrypt and decrypt be efficiently computed?
- How to efficiently generate $p$ and $q$?
- How secure is it?

This cryptosystem, named RSA after its inventors Rivest, Shamir and Adleman, is the most popular public key cryptosystem.

## The RSA cryptosystem

Initial setup:

- $p, q$ two primes
- $n = pq$ and $\varphi(n)$
- $e, d$ such that $ed \equiv 1 \bmod \varphi(n)$

- The $n$ from Bob
- The $e$ from Bob

Encryption:

- A message $m$
- Send $c \equiv m^e \bmod n$ to Bob

Decryption:

- Receive $c$ from Alice or Eve
- Compute $m \equiv c^d \bmod n$

## The RSA cryptosystem

Example.

1. Bob generates:
   - $p = 101$ and $q = 113$
   - $n = 11413$ and $\varphi(n) = 11200$
   - Select a random $e$ and compute both its $gcd(e, \varphi(n))$ and $d = e^{-1}$ using the Extended Euclidian Algorithm (1.51). If the gcd is not one select a different $e$ and retry. Let $e$ be 3533, then $gcd(e, \varphi(n)) = 1$ and $d = 6597$.
2. Bob publishes $n$ and $e$
3. Anybody can use those parameters to encrypt a message and send it to Bob. In particular Alice or Eve can encrypt 9726:
$$9726^{3533} \equiv 5761 \bmod 11413.$$
4. On receiving the message Bob computes
$$5761^{6597} \equiv 9726 \bmod 11413.$$

## Modular exponentiation

The modular exponentiations required to encrypt and decrypt the message can be done efficiently in $\mathcal{O}((\log n)^2 \log d)$ bit operations, using the following algorithm.

Algorithm. (*Square and multiply*)

**Input :** $m$ an integer, $d = (d_{k-1} \dots d_0)_2$ and $n$ two positive integers
**Output:** $x = m^d \bmod n$

```
1 $power \leftarrow 1$;
2 for $i \leftarrow k-1$ to $0$ do
3     $power \leftarrow (power \cdot power) \bmod n$;
4     if $d_i = 1$ then $power \leftarrow (m \cdot power) \bmod n$;
5 end for
6 return $power$
```

## Modular exponentiation

**Example.** Calculate $9726^{3533} \bmod 11413$.

We run the previous algorithm with: $m = 9726$, n=11413 and $d = 3533 = (110111001101)_2$.

| $i$ | $d_i$ | *power* mod 11413 | $i$ | $d_i$ | *power* mod 11423 |
|---|---|---|---|---|---|
| 11 | 1 | $1^2 \cdot 9726 \equiv 9726$ | 5 | 0 | $7783^2 \equiv 6298$ |
| 10 | 1 | $9726^2 \cdot 9726 \equiv 2659$ | 4 | 0 | $6298^2 \equiv 4629$ |
| 9 | 0 | $2659^2 \equiv 5634$ | 3 | 1 | $4629^2 \cdot 9726 \equiv 10185$ |
| 8 | 1 | $5634^2 \cdot 9726 \equiv 9167$ | 2 | 1 | $10185^2 \cdot 9726 \equiv 105$ |
| 7 | 1 | $9167^2 \cdot 9726 \equiv 4958$ | 1 | 0 | $105^2 \equiv 11025$ |
| 6 | 1 | $4958^2 \cdot 9726 \equiv 7783$ | 0 | 1 | $11025^2 \cdot 9726 \equiv 5761$ |

## Faster decryption

Two useful optimizations to the decryption can be applied. The first and most obvious consists in saving $d \bmod \varphi(n)$ such that it is not recomputed at each decryption.

The second idea consists in using the CRT (2.99, 3.145) to speed up the computation. Instead of storing $d \bmod \varphi(n)$ one can save $d \bmod (p-1)$ as well as $d \bmod (q-1)$, recover the “two sub-messages” in $\mathbb{Z}/p\mathbb{Z}$ and $\mathbb{Z}/q\mathbb{Z}$, and combine them over $\mathbb{Z}/n\mathbb{Z}$.

Example. Let $p = 11$, $q = 23$ and $e = 3$. Then $n = 253$, $\varphi(n) = 220$ and $d = 147$. To encrypt $m = 57$ we compute $c = 57^3 \equiv 250 \bmod 253$.

Instead of computing $m \equiv 250^{147} \bmod 253$ we do

$$\begin{cases} 250^{147 \bmod 10} \equiv & 8^7 \equiv & 2 \bmod 11 \\ 250^{147 \bmod 22} \equiv & 20^{15} \equiv & 11 \bmod 23. \end{cases}$$

## Faster decryption

It now suffices to combine the results mod $p$ and $q$ into a single result mod $n$.

Bézout's identity gives $(-2) \cdot 11 + 1 \cdot 23 = 1$. Therefore $1_p$ is mapped into 23 mod 253 and $1_q$ into $-22 \equiv 231$ mod 253.

Hence,

$$\begin{aligned} (2, 11) &= 2 \cdot 1_p + 11 \cdot 1_q \\ &= 2 \ \cdot \ 23 + 11 \ \cdot \ 231 \bmod 253 \\ &\equiv 2587 \bmod 253 \\ &\equiv 57 \bmod 253. \end{aligned}$$

And the plaintext is recovered.

## Generating primes

Which strategy to choose:

- Generate a random integer, pick the next prime
- Generate random integers until one of them is prime

Remark.

- The prime number theorem states that in the range 1-$n$ approximately $n/\ln n$ integers are prime. As we will discuss later, the primes $p$ and $q$ are expected to be about 1024 bits long. Therefore the probability for a random integer between 1 and $2^{1024}$ to be prime is $1/\ln 2^{1024} \approx 1/710$.
- Although a deterministic polynomial time algorithm exists for primality testing (AKS), Monte Carlos algorithms, which are much faster solutions, are often used in practice.

## The Solovay-Strassen primality test

From proposition 3.157 we know that if $n$ is prime then for any $a \not\equiv 0 \bmod n$,

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \bmod n. \tag{3.1}$$

Unfortunately there exist some integers $a$ for which it is also true although $n$ is not prime. It is therefore impossible to derive a deterministic algorithm from this proposition.

On the other hand, we can observe the following property. Let $n$ be composite and $A = \left\{a \;/\; gcd(a, n) = 1 \text{ and } (3.1) \text{ holds}\right\}$. Since $n$ is composite there exists an integer $b$ such that $gcd(b, n) = 1$ and $\left(\frac{b}{n}\right) \not\equiv b^{(n-1)/2}$. For any $a \in A$ we have

$$(ab)^{\frac{n-1}{2}} = a^{\frac{n-1}{2}} b^{\frac{n-1}{2}} = \left(\frac{a}{n}\right) b^{\frac{n-1}{2}} \not\equiv \left(\frac{a}{n}\right)\left(\frac{b}{n}\right) \bmod n.$$

Hence, for any $a \in A$ there is an element coprime to $n$ that does not belong to $A$. It is then possible to construct a Monte Carlo Algorithm that determines whether or not $n$ is prime.

## The Solovay-Strassen primality test

The following algorithm requires $\mathcal{O}(k(\log n)^3)$ operations to test the primality of $n$, $k$ being the number of random elements generated for the test.

Algorithm. (*Solovay-Strassen*)

```
Input  : $n$ an integer, and $k$ the number of tests to run
Output: $n$ is composite or probably prime
1 for $i \leftarrow 1$ to $k$ do
2     $a \leftarrow \mathrm{rand}(2, n-2)$;
3     if $\gcd(a, n) \neq 1$ then return $n$ is composite;
4     $x \leftarrow \left(\frac{a}{n}\right)$;
5     $y \leftarrow a^{(n-1)/2} \bmod n$;
6     if $x \not\equiv y \bmod n$ then return $n$ is composite;
7 end for
8 return $n$ is probably prime
```

## The Miller-Rabin primality test

The Miller-Rabin test is a Monte Carlo Algorithm that determines whether or not an integer is prime.

Let $n \in \mathbb{N}$ be an odd integer. Then $n - 1 = 2^s m$, where $s$ is an integer and $m$ is odd. The integer $n$ passes the *Miller-Rabin test to base a* if either

$$a^m \equiv 1 \bmod n \qquad \text{or} \qquad a^{2^j m} \equiv -1 \bmod n$$

for some $j$ with $0 \leq j \leq s - 1$.

To see it, observe that if $n$ is prime then $x^2 \equiv 1 \bmod n$ has only two solutions: $+1$ and $-1$. Moreover Fermat's little theorem (2.95) applies and $a^{n-1} \equiv 1 \bmod n$.

Therefore taking the square root of $a^{n-1}$ yields 1 or $-1$. On $-1$ the second congruence holds. If this is 1 then the square root can be taken again until it is either $-1$ or only $m$ is left. Hence one of the two congruences holds.

## The Miller-Rabin primality test

Finally the contrapositive states that if neither of the congruences holds then $n$ is composite.

Noticing the two following points we can now derive a probabilistic algorithm which returns whether an integer is composite or probably prime.

- If $n$ is prime and $1 < a < n$, then $n$ passes Miller's test to base $a$.
- If $n$ is composite, then there are fewer than $n/4$ bases $a$ with $1 < a < n$ such that $n$ passes Miller's test to base $a$.

## The Miller-Rabin primality test

The Monte Carlo algorithm now randomly selects $k$ bases $a$ and performs the Miller-Rabin test.

- If $n$ fails the test for any of the bases used, the algorithm will return "true" ($n$ is composite).
- If $n$ passes each test, the answer is still unknown. Nevertheless, the algorithm will return "false" ($n$ is probably prime).

The probability that $n$ is composite and still passes the test each of the $k$ times is

$$p_k = \frac{1}{4^k}.$$

For instance if $k = 30$ tests are performed, $p_k < 10^{-18}$. It is almost certain that a number that the algorithm returns as prime actually is prime.

## The Miller-Rabin primality test

**Algorithm.** (*Miller-Rabin*)

```
Input  : $n$ an odd integer, and $k$ the number of tests to run
Output : $n$ is composite or probably prime
$m \leftarrow (n-1)/2$; $s \leftarrow 1$;
while $2|m$ do  $m \leftarrow m/2$; $s \leftarrow s+1$;
for $i \leftarrow 1$ to $k$ do
    $a \leftarrow \text{rand}(2, n-2)$;
    if $\gcd(a, n) \neq 1$ then return $n$ is composite;
    $a \leftarrow a^m \bmod n$;
    if $a = \pm 1$ then continue;
    for $j \leftarrow 1$ to $s-1$ do
        $a \leftarrow a^2 \bmod n$;
        if $a \equiv 1 \bmod n$ then return $n$ is composite;
        if $a \equiv -1 \bmod n$ then b$\leftarrow$ 1 ; break;
    end for
    if $b=1$ then continue else return $n$ is composite;
end for
return $n$ is probably prime
```

## Testing RSA security

The last question that remains to be answered is related to the security of RSA. The RSA cryptosystem can be viewed as having three secret parameters: $p, q$ and $d$.

If $n$ and $\varphi(n)$ are known, then $p$ and $q$ can be efficiently recovered. Note that

$$n - \varphi(n) + 1 = pq - (p-1)(q-1) + 1 = p + q.$$

Since we know $pq$ and $p+q$, $p$ and $q$ are the roots of the quadratic equation $X^2 - (n - \varphi(n) + 1)X + n$. Hence

$$p, q = \frac{n - \varphi(n) + 1 \pm \sqrt{(n - \varphi(n) + 1)^2 - 4n}}{2}.$$

Said otherwise, if $\varphi(n)$ can be computed then $n$ can be factorised. Since factorizing $n$ is believed to be hard there should be no way of efficiently compute $\varphi(n)$.

## Testing RSA security

Suppose that both $e$ and $d$ are known. Then $n$ can be efficiently factorised.

Since $de \equiv 1 \bmod \varphi(n)$, for any $a$ coprime to $n$, $a^{de-1} \equiv 1 \bmod n$.

The idea is now to select some random $a$ coprime to $n$, and apply the strategy described on slide 3.153. Note that if $a$ and $n$ are not coprime then $\gcd(a, n)$ is factor of $n$. Therefore lets assume their gcd to be 1.

We start by writing $ed - 1$ as $2^s m$ and then define $b_0 = a^m$ and $b_{i+1} \equiv b_i^2 \bmod n$. As we expect to find the order of $a$ we want $b_{i+1} \equiv 1 \bmod n$, while $b_i \not\equiv 1 \bmod n$. Moreover if $b_i \equiv -1 \bmod n$ then the factors are trivial. Therefore our aim is to find an $a$ such that $b_i \not\equiv \pm 1$ and $b_{i+1} \equiv 1 \bmod n$. In this case, $\gcd(b_i - 1, n)$ and $\gcd(b_i + 1, n)$ are non-trivial factors of $n$.

Hence finding $d$ should be hard since it allows to factorize $n$.

## The RSA problem

From the previous discussion it appears that the RSA cryptosystem relies on the hardness of factoring large composite integers. But more precisely it is the hardness of determining $\varphi(n)$ when only $n$ is known without its prime decomposition. The RSA problem can be formally stated as follows.

**Problem** (RSA problem)

Let $n$ be a large integer and $e > 0$ be coprime to $\varphi(n)$. Given $y$ in $\mathrm{U}(\mathbb{Z}/n\mathbb{Z})$, compute $y^{1/e} \bmod n$, i.e. find $x$ such that $x^e \equiv y \bmod n$.

Although factoring $\varphi(n)$ or computing $d$ solves the RSA problem there is no proof that no other way of solving it exists. Therefore it cannot be concluded that the RSA problem is as hard as factoring. Indeed it may be that the RSA problem can be solved in polynomial time even though the factoring problem cannot.

## Factoring integers

Complexity of a few factorization algorithms for $n$ a $k$-bit integer:

| Algorithm | Complexity |
|---|---|
| Trial division | $\mathcal{O}\left(2^{k/2}/k)\right)$ |
| Pollard-$\rho$ | $\mathcal{O}\left(\sqrt[4]{n}\right)$ |
| ECM | $L_p\left[1/2, \sqrt{2}\right]$ |
| GNFS | $L_n\left[1/3, \sqrt[3]{64/9}\right]$ |

The $L_n(\alpha, c)$ function is defined by

$$L_n(\alpha, c) = e^{\left(c+o(1)\right)\left((\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}\right)}.$$

## Factoring integers

Given a large random integer $N$, the probability for $N$ to be divisible by 2 is 1/2; by 3, 1/3; by 5, 1/5 etc. One can deduce that about 88% of integers have a factor smaller than 100 and 92% a factor smaller than 1000.

Therefore, despite its exponential complexity, trial division is used in almost all factoring programs. All the small factors are first removed before more advanced strategies are employed to totally factorize $N$.

In practice, trial division is implemented through a large table containing all the primes, or alternatively the difference between two consecutive primes, up to 10 million. Then even for a 1000 digit long integer it only takes a few seconds to perform all the trial divisions and ensure that $N$ is free of any small factor.

## Factoring integers

Another simple idea in order to remove small factors consists in computing $\gcd(n, P)$ where $P$ corresponds to the product of all the prime numbers below a given bound $B$. Compared to trial division this strategy seems appealing since computing a gcd can be done in polynomial time. In practice, this method is much more efficient when considering primes below 1000 but it becomes extremely slow when checking prime factors of size around one million.

In the first case the product of all the primes is about 1,400 bits while in the second case it is approximately 1,500,000! Computing the gcd of an integer around $2^{3072}$ and $P$, then takes much longer.

This simple example highlights how practical cases can highly diverge from the theoretical asymptotic analysis.

## Pollard's Rho Algorithm

We now introduce an example of a more sophisticated factoring scheme. It is asymptotically faster than trial factorization and can be used when small numbers have been eliminated as possible factors.

Let $n$ be a composite integer with an unknown prime factor $p \leq \sqrt{n}$. Define the function

$$f\colon \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}, \qquad f(x) = x^2 + 1 \bmod n$$

(other functions $f\colon \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ can be used). We now recursively define a sequence $(x_k)$ by

$$x_0 = 2, \qquad x_{k+1} = f(x_k), \qquad k \in \mathbb{N}.$$

Since there are exactly $n$ elements in $\mathbb{Z}/n\mathbb{Z}$, the sequence must at some point produce a repeated value and enter a cycle.

## Pollard's Rho Algorithm

We then **hope** that the cycle contains two or more elements with the same remainder modulo $p$, i.e., that we can find $x_i$ and $x_j$, $i \neq j$, in the cycle such that

$$x_i \equiv x_j \bmod p.$$

If that is the case, then $x_i - x_j$ is divisible by $p$ and $\gcd(x_i - x_j, n)$ gives a factor of $n$.

In summary, when testing all of the $x_i$ and $x_j$ of the cycle, this GCD can evaluate as follows:

$$\gcd(x_i - x_j, n) = \begin{cases} n & \text{if } x_i = x_j, \\ 1 & \text{if } x_i \not\equiv x_j \bmod p \text{ for all factors } p \text{ of } n, \\ t & \text{if } x_i \equiv x_j \bmod p, \text{ where } p \mid t \text{ and } t \mid n. \end{cases}$$

## Pollard's Rho Algorithm

The algorithm now uses the following method to evaluate pairs $x_i, x_j$ in the cycle: two sequences $(x_k)$ and $(y_k)$ are defined,

$$x_0 = 2, \quad x_{k+1} = f(x_k) \qquad \text{and} \qquad y_0 = 2, \quad y_{k+1} = f(f(y_k)).$$

The sequences $(x_k)$ traverses the cycle normally, while the sequence $(y_k)$ traverses the cycle in double steps. This is intended to be an efficient manner of generating "random" pairs $(x_i, x_j)$. For each pair, $\gcd(x_i - x_j, n)$ is evaluated.

Example. Suppose that we want to factor the number $n = 8051$. We start with $x_0 = 2$ and set $x_{k+1} = x_k^2 + 1 \bmod 8051$. We obtain the sequence

$$\begin{aligned}(x_i) = (&2,\ 5,\ 26,\ 677,\ \mathbf{7474},\ 2839,\ 871,\ 1848,\ 1481,\ 3490,\ 6989,\\ &705,\ 5915,\ 5631,\ 3324,\ 3005,\ 4855,\ 5749,\ 1647,\ \mathbf{7474},\ 2839, \ldots\end{aligned}$$

and we have found a cycle starting at $x_4 = 7474$.

## Pollard's Rho Algorithm

In practice, we simply generate the sequences $(x_k)$ and $(y_k)$ and evaluate the GCDs:

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_k$ | 2 | 5 | 26 | 677 | 7474 | 2839 |
| $y_k$ | 2 | 26 | 7474 | 871 | 1481 | 6989 |
| $\gcd(x_k - y_k, n)$ | 8051 | 1 | 1 | 97 | 1 | 83 |
| $x_k$ | 871 | 1848 | 1481 | 3490 | 6989 | 705 |
| $y_k$ | 5915 | 3324 | 4855 | 1647 | 2839 | 1848 |
| $\gcd(x_k - y_k, n)$ | 97 | 1 | 1 | 97 | 83 | 1 |
| $x_k$ | 5915 | 5631 | 3324 | 3005 | 4855 | 5749 |
| $y_k$ | 3490 | 705 | 5631 | 3005 | 5749 | 7474 |
| $\gcd(x_k - y_k, n)$ | 97 | 1 | 1 | 8051 | 1 | 1 |

Even before the cycle is entered by $(x_k)$, a factor $p = 97$ of $n = 8051$ is found.

## Pollard's Rho Algorithm

Algorithm. (*Pollard-$\rho$ – Factorization*)

**Input:** $n$, a composite integer, $f(x) = x^2 + 1 \bmod n$.
**Output:** $d$ a non-trivial factor of $n$, or failure.

```
a ← 2; b ← 2;
repeat
    a ← f(a); b ← f(f(b));
    d ← gcd(a − b, n);
until d ≠ 1;
if d = n then
    return failure
else
    return d
end if
```

## Complexity of Pollard's Rho Algorithm

The Pollard Rho algorithm derives its name from the shape of the sequence $(x_k)$. At some point in the sequence, $x_k \equiv x_{k+T} \bmod p$ for some $T > 0$ and the sequence can be represented as a cycle from that point onwards - this is the circle of the letter $\rho$. The sequence terms $x_0, x_1, \dots x_{k-1}$ then form the "tail" of $\rho$.

Remark.

- The role of $f$ is to "randomly" select numbers in $\mathbb{Z}/n\mathbb{Z}$. Its precise form is not essential, but it should be a polynomial for
$$f(f(x) \bmod n) \bmod n = f(f(x)) \bmod n$$
when calculating the sequence $(y_k)$.
- It is not guaranteed that the Pollard Rho algorithm actually will be successful - it could happen that all of the $x_i$ in the cycle have distinct remainders modulo $p$. In that case, a different starting point $x_0$ should be chosen and the algorithm run once more.

## Complexity of Pollard's Rho Algorithm

We now attempt to make a rough estimate of the average time complexity of the algorithm. Let us ignore the specifics and suppose that the algorithm simply selects random numbers $x_i, x_j \in \mathbb{Z}/n\mathbb{Z}$ for comparison of their remainders.

Suppose that any given number between 0 and $n$ has an equal probability $1/p$ of having a remainder $m$ modulo $p$, $0 \leq m < p$:

$$P[x_k \bmod p = m] = \frac{1}{p} \qquad \text{for all } m = 0, \dots, p-1 \text{ and all } k \in \mathbb{N}.$$

Suppose that for any two $x_i$ and $x_j$, $i \neq j$, these probabilities are independent. Then the probability that $x_i$ and $x_j$ have different remainders is

$$P[x_i \not\equiv x_j \bmod p] = \frac{p-1}{p},$$

## Complexity of Pollard's Rho Algorithm

Suppose that $x_0, \dots, x_{k-1}$ have distinct remainders modulo $p$, then the probability of $x_k$ to have a remainder different from $x_0, \dots, x_{k-1}$ is $\frac{p-k}{p}$.

Then (assuming the independence of the value of the remainder) the probability that a group of $k$ numbers has distinct remainders mod $p$ is

$$\begin{aligned} P_k &:= P[x_i \not\equiv x_j \bmod p,\ 0 \leq i < j \leq k] \\ &= \prod_{l=0}^{k} \frac{p-l}{p} = \frac{p!}{(p-k-1)!p^k}. \end{aligned}$$

It can be shown that $1 - P_k < 1/2$ if $k > 1.177\sqrt{p}$.

This indicates that the average-case complexity of Pollard's Rho algorithm should be

$$\mathcal{O}(\sqrt{p}) = \mathcal{O}(\sqrt[4]{n}).$$

## Factoring the RSA modulus

In the RSA case $n$ is known to be product of two large primes. Therefore the algorithm of choice is the GNFS. Then applying the strategy described on slide 1.70 we set

$$2^{128} = e^{\sqrt[3]{\frac{64}{9}}(\ln n)^{1/3}(\ln\ln n)^{2/3}},$$

which gives approximately $n = 7.65 \cdot 10^{763}$. In terms of bit length, $n$ should be about 2500 bits long. In practice, this is rounded up to 3072 bits (2048+1024), and a bit length of 3072 is considered "secure" as it corresponds to 128 bits security.

As of 2014, the largest product of two large primes officially factorized occurred in the breaking of RSA-768 (a 768 bits RSA modulus):

1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786
469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479
274737794080665351419597459856902143413
$=$
3347807169895689878604416984821269081770479498371376856891243138898288379387800228761471165253174
3087737814467999489
$\times$
3674604366679959042824463379962795263227915816434308764267603228381573966651127923337341714339681
0270092798736308917

## Attacks on RSA

Strategy for short messages and small $e$:

- A message $m < n^{1/e}$
- The ciphertext is $c \equiv m^e \bmod n$
- The encryption does not require any modular reduction
- Over the integers $c$ is also $m^e$
- Solve $c^{1/e}$ over the integers recovers $m$

Typical use: encrypt a 128 bits long secret key using RSA

## Attacks on RSA

Strategy for short messages:

- A message $m$ of less than about $10^{17}$ bits
- The ciphertext is $c = m^e \bmod n$
- Compute and store in a table $cx^{-e} \bmod n$, for all $1 \leq x \leq 10^9$
- Compute $y^e \bmod n$, for all $1 \leq y \leq 10^9$, and test for a collision
- If a collision is found then $c = (xy)^e \bmod n$
- If $m \leq 10^{17}$ it is likely that such $x$ and $y$ exist

## Attacks on RSA

Strategy for small $e$:

- A message $m$ sent to $i \geq e$ persons using the keys $\langle n_i, e \rangle$
- If $\gcd(n_k, n_j) \neq 1$ then one of the $n_i$ can be factorized
- Otherwise set $n = \prod_i n_i$
- Use the CRT over all the ciphertext $c_i = m^e \bmod n_i$, to compute $c \equiv m^e \bmod n$
- As $m < \min_i(n_i)$ and $i \geq e$, then $m^e < n$
- Finally $m = c^{1/e}$ can be computed in $\mathbb{N}$

## A more secure RSA

Current version of RSA:

- Referred to as “textbook RSA”
- Insecure but simple to understand

PKCS #1 v1.5:

- Randomly pad the message before encrypting
- Assumed to be CPA secure
- No proof based on the RSA problem
- Not CCA secure
- Still widely used for backward-compatibility reasons

## Toward a secure RSA

Optimal Asymmetric Encryption Padding (RSA-OAEP):

- Due to Bellare and Rogaway
- Standardized as PKCS #1 v2
- Similar to feistel network in the construction
- Proved to be CCA secure

Notations:

- Concatenation of two bit strings $a$ and $b$: $a \| b$
- Repetition of a bit $b$, $d$ times: $b^d$
- Random oracle are informaly defined on slide 2.103

## RSA-OAEP

Generate: set $p, q, n, e$, and $d$, and choose two random oracles $G : \{0,1\}^{2l} \to \{0,1\}^{2l}$ and $H : \{0,1\}^{2l} \to \{0,1\}^{2l}$, for $l \in \mathbb{N}$

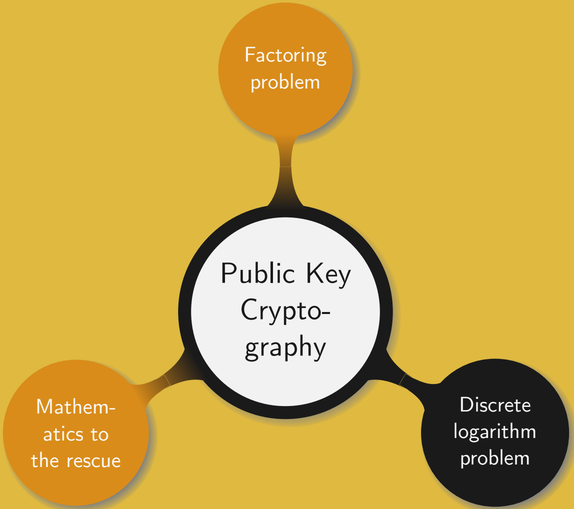
Encrypt: a message $m \in \{0,1\}^{l}$

1. Pick a random $r \in \{0,1\}^{2l}$
2. Set $m' = m \| 0^{l}$
3. Compute $m'' = \left(G(r) \oplus m'\right) \| \Big(r \oplus H\left(G(r) \oplus m'\right)\Big)$
4. Define the ciphertext as $c = (m'')^{e} \bmod n$.

Decrypt: a ciphertext $c$

1. Compute $m'' = c^{d} \bmod n$.
2. Parse $m''$ as $m_1 \| m_2$, with $m_1, m_2$ of size $2l$
3. Recover $r = H(m_1) \oplus m_2$
4. Recover $m' = m_1 \oplus G(r)$
5. If the $l$ last bits are not $0^{l}$ output error $\lightning$
6. Otherwise output $m$, the first $l$ bits of $m'$

## Chapter organisation



## The Discrete Logarithm Problem

After investigating the RSA problem (3.185) we now turn our attention to another hard problem from number theory.

**Problem** (Discrete Logarithm Problem (DLP))

Let $\mathbb{F}_q$ be a finite field, with $q = p^n$, for a positive integer $n$. Given $\alpha$ a generator of $G$, a subgroup of $\mathbb{F}_q^*$, and $\beta \in G$, find $x$ such that $\beta = \alpha^x$ in $\mathbb{F}_q$.

Note that $x$ is unique only up to congruence mod $|G|$, therefore $x$ is usually restricted to $0 \leq x < \mathrm{ord}_{\mathbb{F}_q^*}(\alpha)$.

## The Discrete Logarithm Problem

Example. For $p = 13$ and $n = 1$ the field of concern is $\mathbb{Z}/13\mathbb{Z}$. The multiplicative group $\mathrm{U}(\mathbb{Z}/13\mathbb{Z})$ has order 12 and as such has a subgroup of order 6 (Lagrange's theorem (3.147)).

From example 3.144, 2 has order 12 and is a generator of $\mathrm{U}(\mathbb{Z}/13\mathbb{Z})$. Therefore 4 generates a subgroup of order 6, namely

$$G = \{4, 3, 12, 9, 10, 1\} .$$

Example of DLP in $G$: find $x$ such that $4^x \equiv 9 \bmod 13$.
Clearly 4 is a solution, but also 10, 16, 22...However, restricting $x$ to the range 0 – 6 makes it unique.

## Pollard's Rho Algorithm

In the previous section Pollard's Rho algorithm was investigated as a way to solve the factorization problem. Since Factorization and Discrete Logarithm have much in common Pollard's Rho algorithm can be adjusted to this new context. We now present its details.

Let $\alpha$ be a generator of a group $G$ of prime order $p$. Any element of $G$ can be written $\alpha^a\beta^b$ for some $a, b \in \mathbb{N}$ and $\beta \in G$.

Assuming two integers $x$ and $y$ such that $x \equiv y \bmod p$ can be found, then there exist $a_1, b_1$ such that $x \bmod p$ can be written $\alpha^{a_1}\beta^{b_1}$, and $a_2, b_2$ such that $\alpha^{a_2}\beta^{b_2} \equiv y \bmod p$.

Rewriting $x \equiv y \bmod p$ as $\alpha^{a_1}\beta^{b_1} \equiv \alpha^{a_2}\beta^{b_2} \bmod p$ yields

$$\beta^{b_1-b_2} \equiv \alpha^{a_2-a_1} \bmod p.$$

Taking the $\log_\alpha$ on both sides leads to

$$(b_1 - b_2)\log_\alpha \beta = a_2 - a_1 \bmod p.$$

## Pollard's Rho Algorithm

As long as $p \nmid (b_1 - b_2)$ we get

$$\log_\alpha \beta = \frac{a_2 - a_1}{b_1 - b_2}.$$

Therefore the goal of Pollard's Rho algorithm is to find $x$ and $y$ with $x \equiv y \bmod p$. This is achieved by considering three partitions $S_1$, $S_2$ and $S_3$ of $G$ of approximately the same size, based on an easily testable property, and defining three functions $f$, $g$ and $h$.

$$f(x) = \begin{cases} \beta x & x \in S_1 \\ x^2 & x \in S_2 \\ \alpha x & x \in S_3 \end{cases}$$

$$g(a,x) = \begin{cases} a & x \in S_1 \\ 2a \bmod p & x \in S_2 \\ a+1 \bmod p & x \in S_3 \end{cases} \qquad h(b,x) = \begin{cases} b+1 \bmod p & x \in S_1 \\ 2b \bmod p & x \in S_2 \\ b & x \in S_3 \end{cases}$$

## Pollard's Rho Algorithm

Starting with two elements $x = y = 1$, $f$ is iteratively applied, once to $x$ and twice to $y$. Since $G$ is a cyclic group repeatedly applying $f$ to $x$ and $y$ will result in a collision at some stage.

The function $f$, $g$ and $h$ are defined such as the progress of $x$ and $y$ appears "random", while $y$ goes twice as fast as $x$. Then by the birthday paradox (4.231) a collision can be expected in time $\sqrt{p}$, since $p$ is the order of the group $G$.

Remark. The cyclic group $G$ is taken as generic and no further assumption is made. This means that Pollard's Rho method applies to any group $G$ of prime order $p$.

## Pollard's Rho Algorithm

Algorithm. (*Pollard-$\rho$ – Discrete Logarithm*)

**Input** : $\alpha$ a generator of $G$, a group of prime order $p$ and $\beta \in G$, $f$, $g$ and $h$ three functions.

**Output:** $\log_\alpha \beta$, or failure.

```
1 $a_1 \leftarrow 0$; $b_1 \leftarrow 0$; $x \leftarrow 1$; $a_2 \leftarrow 0$; $b_2 \leftarrow 0$; $y \leftarrow 1$;
2 **repeat**
3     $a_1 \leftarrow g(a_1, x)$; $b_1 \leftarrow h(b_1, x)$;
4     $x \leftarrow f(x)$;
5     $a_2 \leftarrow g(g(a_2, y), f(y))$; $b_2 \leftarrow h(h(b_2, y), f(y))$;
6     $y \leftarrow f(f(y))$;
7 **until** $x \equiv y \bmod p$;
8 $r \leftarrow b_1 - b_2$;
9 **if** $r \neq 0$ **then** **return** $r^{-1}(a_2 - a_1) \bmod p$;
10 **else return** failed;
```

## Pollard's Rho Algorithm

Example. Let $\alpha = 2$ be a generator of $G$, the subgroup of order 191 of $\mathbb{Z}_{383}^*$. Let $\beta = 228$. Partition $G$ into $S_1 = \{x \in G | x \equiv 1 \bmod 3\}$, $S_2 = \{x \in G | x \equiv 0 \bmod 3\}$ and $S_3 = \{x \in G | x \equiv 2 \bmod 3\}$.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x$ | 228 | 279 | 92 | 184 | 205 | 14 | 28 |
| $a_1$ | 0 | 0 | 0 | 1 | 1 | 1 | 2 |
| $b_1$ | 1 | 2 | 4 | 4 | 5 | 6 | 6 |
| $y$ | 279 | 184 | 14 | 256 | 304 | 121 | 144 |
| $a_2$ | 0 | 1 | 1 | 2 | 3 | 6 | 12 |
| $b_2$ | 2 | 4 | 6 | 7 | 8 | 18 | 38 |
| $x$ | 256 | 152 | 304 | 372 | 121 | 12 | **144** |
| $a_1$ | 2 | 2 | 3 | 3 | 6 | 6 | 12 |
| $b_1$ | 7 | 8 | 8 | 9 | 18 | 19 | 38 |
| $y$ | 235 | 72 | 14 | 256 | 304 | 121 | **144** |
| $a_2$ | 48 | 48 | 96 | 97 | 98 | 5 | 10 |
| $b_2$ | 152 | 154 | 118 | 119 | 120 | 51 | 104 |

Then compute $(38-104)^{-1}(10-12) \equiv 110 \bmod 191$. Hence in $\mathbb{Z}_{383}^*$ $\log_2(228) = 110$.

## Polhig-Hellman Algorithm

Although slightly more advanced Polhig-Hellman Algorithm is interesting in the sense that it takes advantage of the structure of the prime $p$. In fact it was noticed that if $p-1$, the order of the multiplicative group of $\mathbb{Z}_p$, is featuring many small primes then the Discrete Logarithm Problem can be solved using the Chinese Remainder Theorem.

Let $p-1 = q_1^{e_1} q_2^{e_2} \dots q_r^{e_r}$, $r \in \mathbb{N}$. If $x = \log_\alpha \beta$ then it suffices to determine $x_i = x \bmod q_i^{e_i}$ for $1 \le i \le r$ and then use the Chinese Remainder Theorem in order to recover $x$. Therefore it only remains to compute all the $x_i$. This can be efficiently achieved at the cost of some mathematical technicalities.

## More on the Discrete Logarithm Problem

Remark. A common mathematical strategy consists in transposing a difficult problem over a given structure into an easier one over a similar structure. Following this idea one could think of solving a hard Discrete Logarithm Problem into an isomorphic group and then map it back to the original group.

In particular since the Discrete Logarithm Problem is easy to solve in the additive group $\mathbb{Z}_n$, $n \in \mathbb{N}$, it is possible to map the multiplicative group of $\mathbb{Z}_p$ into the additive group $\mathbb{Z}_{p-1}$. Solving the problem in this simpler group and mapping back the solution to $\mathbb{Z}_p$ seems to be a very attractive solution.

The major problem with this approach is finding the map. In fact such a map would have to be built element by element, which would be time consuming. As a result this solution is not applicable in practice.

## The DLP in cryptography

It is simple to see that the DLP is the inverse operation of the modular exponentiation, which can be very efficiently computed (3.172). However solving the DLP is not an easy task if the group is carefully chosen.

For instance as we will study in chapter 8, in groups having only a very basic algebraic structure the best algorithm available is the Pollard's Rho algorithm.

For more common groups over finite fields the best algorithms have a sub-exponential complexity similar to the one of the GNFS. Therefore in a cryptographic context, that is for the DLP to be intractable, the group is expected to have order larger than $2^{3072}$.

We now present several cryptographic protocols based on the hardness of solving the DLP.

## Diffie-Hellman key exchange

Alice and Bob publicly agree on some parameters:

$G$ a group of order $p$

$\alpha$ a generator of $G$

Both Alice and Bob generate a random secret:

Choose a random element $x$ in $G$

Choose a random element $y$ in $G$

Alice and Bob send each other $\alpha^{secret}$:

- $x$ in $G$ and $\alpha^y$
- $\alpha^{yx}$

- $y$ in $G$ and $\alpha^x$
- $\alpha^{xy}$

## Diffie-Hellman problems

Clearly solving the DLP implies breaking the Diffie-Hellman key exchange protocol. However in order to determine $\alpha^{xy}$ it might not be necessary to solve the DLP, but only to solve the so called Computational Diffie-Hellman problem.

**Problem** (Diffie-Hellman problems)

Let $G$ be a group of prime order $p$ and $\alpha$ be a generator of $G$.

1. *Computational Diffie-Hellman (CDH):* given $\alpha^x$ and $\alpha^y$, for some unknown integers $x$ and $y$, compute $\alpha^{xy}$.
2. *Decisional Diffie-Hellman (DDH):* given $\alpha^x$ and $\alpha^y$, decide whether or not some $c \in G$ is equal to $\alpha^{xy}$.

While solving the DLP implies solving the CDH problem, it is not known whether or not solving the CDH problem solves the DLP.

At present no method to solve CDH from DDH is known, and in fact in some groups DDH is efficiently solved while CDH remains hard.

## Elgamal cryptosystem

Initial setup:

- $G$ a group of prime order $p$
- $\alpha$ a generator of $G$
- $x$ a secret integer

- $G$ from Bob
- $\alpha$ from Bob
- $\beta \equiv \alpha^x \bmod p$ from Bob

Encryption:

- A message $m$
- $r \equiv \alpha^k \bmod p$ for some random integer $k$
- Compute $t \equiv \beta^k m \bmod p$
- Send $c = \langle r, t \rangle$ to Bob

Decryption:

- Receive $c$ from Alice or Eve
- Compute $tr^{-x} \equiv \beta^k m \left(\alpha^k\right)^{-x} \equiv \left(\alpha^x\right)^k m\alpha^{-xk} \equiv m \bmod p$

## Elgamal and CDH

**Proposition**

Solving the CDH problem is equivalent to breaking Elgamal.

*Proof.* Assume we can decrypt any Elgamal ciphertext and we are given $\alpha^a$ and $\alpha^b$. First set $\beta$ to $\alpha^a$, i.e. $a$ is the secret key, and $r$ to $\alpha^b$. Then choosing a non zero value for $t$ we get

$$m \equiv tr^{-a} \equiv t\alpha^{-ab}.$$

Hence $tm^{-1} \equiv \alpha^{ab} \bmod p$ solves the CDH problem (3.216).

Conversely assume we can solve the CDH problem and are given the Elgamal ciphertext $\langle r, t \rangle$. Then we can determine $\alpha^{xk} \bmod p$, and since $m$ is $t\alpha^{-xk}$ we can recover the plaintext. $\square$

## CCA on Elgamal

Given an Elgamal ciphertext $c = \langle r, t \rangle$, the goal is to recover the plaintext corresponding to $c$ by constructing another ciphertext $c' = \langle r', t' \rangle$ and getting it decrypted by a “decryption oracle”.

Constructing a chosen ciphertext $c''$:

- By definition $c = \langle r, t \rangle = \langle \alpha^k, \beta^k m \rangle$
- For a message $m'$, compute $c'' = \langle r\alpha^{k'}, t\beta^{k'} m' \rangle$
- Submitting $c''$ to the decryption oracle, $m''$ is returned

Recovering $m$ from $m''$:

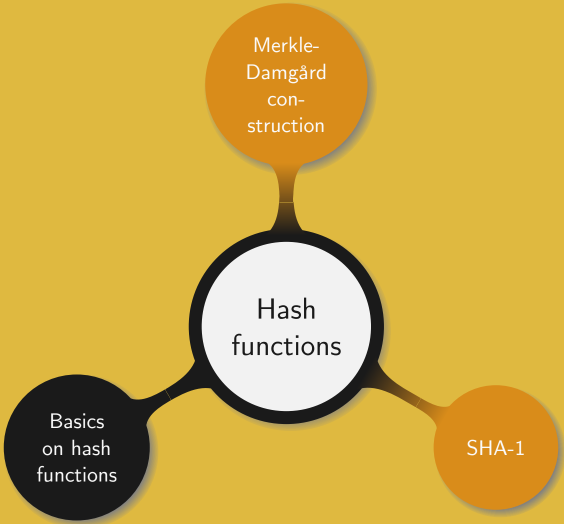
- In the group $G$, observe that $c''$:
  - $r\alpha^{k''} = \alpha^{k+k'} = \alpha^{k''}$
  - $t\beta^{k''} m' = \beta^k m \beta^{k'} m' = mm'\beta^{k+k'}$
- Then $c''$ is the ciphertext corresponding to $m'' = mm'$
- Finally compute $m = m'' m'^{-1}$

## Key points

- What is the order of a group, of an element?
- How to efficiently perform primality testing?
- Which version of RSA is secure? Describe it
- When to use Diffie-Hellman key exchange protocol?
- Describe Elgamal

## 4. Hash functions

## Chapter organisation



## Data integrity

Components of modern cryptography:

- Confidentiality
- Authentication
- Data integrity
- Non-repudiation

Common setup for data integrity:

- Insecure environment
- Unencrypted data
- Data must not be altered

Example.

- Files in an OS
- Software to be downloaded or installed from internet

## Fingerprint

Simple high-level idea:

- Construct a short fingerprint of the data
- Store the fingerprint in a secure place
- Recompute and compare the fingerprint on a regular basis
  - Fingerprint is changed: data was altered
  - Fingerprint is unchanged: data was not altered

Construction goals:

- The fingerprint must be a few hundreds of bits long
- A tiny change in that data radically impacts the fingerprint
- It is impossible to alter the data without totally changing the fingerprint

## Hash function

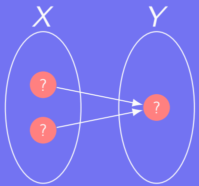
**Definition** (Hash function)

A *hash function* is a function $h$ that verifies the following properties:

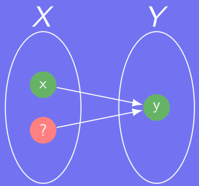
- i Efficiently computed for any input.
- ii *Pre-image resistant*: given $y$ it is computationally infeasible to find $x$ such that $h(x) = y$.
- iii *Second pre-image resistant*: given $x$, it is computationally infeasible to find $x' \neq x$ with $h(x) = h(x')$.
- iv *Collision resistant*: it is computationally infeasible to find $x$ and $x'$ with $x' \neq x$ and $h(x) = h(x')$.

The output of a hash function is called *message digest* or *digest*.

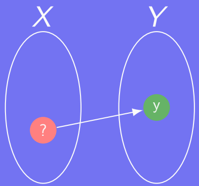
## Remarks



Collision Resistance



2nd Pre-image Resistance



Pre-image Resistance

By definition a hash function is a one-way function since the definition of pre-image resistant is the same as the one of a one-way function.

The collision resistance is the most general of the three resistance properties. Roughly speaking collision resistance implies second pre-image resistance, which implies pre-image resistance.

## Applications

Remark. A hash function takes inputs of any size and output values of fixed sizes, independently of the input.

Applications requiring the resistance properties:

- Pre-image resistant: password
- Second pre-image resistant: virus
- Collision resistance: signature

## The DLP hash function

We present a first example of a hash function due to Chaum, van Heijst and Pfitzmann. Although it satisfies conditions (ii), (iii), and (iv) of a hash function, it is too slow to be used in practice.

Let $p$ be a prime such that $q = \frac{p-1}{2}$ is also a prime and choose $\alpha$ and $\beta$ two generators of $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$. We write any $x$ in $\mathbb{Z}/q^2\mathbb{Z}$ as $x_0 + x_1 q$ with $0 \leq x_0, x_1 \leq q-1$, and define $h$ from $\mathbb{Z}/q^2\mathbb{Z}$ into $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$ by

$$h(x) = \alpha^{x_0}\beta^{x_1} \bmod p.$$

We will now show that $h$ is collision resistant, by proving that finding a collision means solving the DLP. To prove this result we will need to recall a result on the number of solutions of a modular equation.

## The DLP hash function

**Proposition**

If two values $x \neq x'$ with $h(x) = h(x')$ are known then the discrete logarithm of $\beta$ in base $\alpha$ can be efficiently computed.

In order to prove this result we recall the following lemma.

**Lemma**

Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$ and $d = \gcd(a, m)$. The linear congruence $ax \equiv b \bmod m$ has a solution if and only if $d \mid b$. In that case, it has $d$ solutions that are mutually incongruent mod $m$.

## The DLP hash function

Proof. Suppose $h(x) = h(x')$, with $x = x_0 + x_1 q$ and $x' = x'_0 + x'_1 q$. Since $\alpha$ is a generator of $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$, $\beta = \alpha^a$ for some integer $a$ and

$$\alpha^{x_0 + a x_1} \equiv \alpha^{x'_0 + a x'_1} \bmod p.$$

From corollary 3.152, $a(x_1 - x'_1) \equiv x'_0 - x_0 \bmod (p-1)$. To solve this equation for $a$ we see that if $x_1 = x'_1$ then $x_0 = x'_0$ and $x = x' \lightning$.

Therefore we assume $x_1 \neq x'_1$, and find $d = \gcd(x_1 - x'_1, p-1)$ incongruent solutions (lemma 4.229). But as $q = \frac{p-1}{2}$ is prime, the only factors of $p-1$ are $1, 2, q$, and $p-1$. Also note that $0 \leq x_1, x'_1 \leq q-1$ implies $-(q-1) \leq x_1 - x'_1 \leq q-1$. And since $x_1 - x'_1 \not\equiv 0 \bmod (p-1)$ it means that $d$ is either 1 or 2.

Thus it suffices to test the two solutions to determine $a$. Hence finding $x \neq x'$ with $h(x) = h(x')$ implies solving the DLP. $\square$

## The birthday paradox

As we have already mentioned the birthdays paradox on several occasions (2.109, 3.209) we now present some more details on this “birthday attack”.

The essence of the birthday paradox can be expressed by considering the birthdays of 23 persons. We first assume the birthdays to be independent and equiprobable. If those 23 people all have a different birthday, it means that the second person has 364/365 chance of not sharing a birthday with the first one. Then for the third one the probability is 363/365 and so on until the twenty-third whose probability is 343/365. Therefore the probability of at least two sharing the same birthday is

$$1 - \prod_{i=1}^{22} \frac{365 - i}{365} = 0.507 > \frac{1}{2}.$$

## The birthday paradox

Suppose we now have a large number of objects $n$, that are randomly chosen with replacement by two groups of $r$ persons each. The probability of someone in the first group choosing the same object as someone in the second group can be approximated by $1 - e^{-r^2/n}$.[1] And the probability of $i$ matches is $\left(\frac{r^2}{n}\right)^i \frac{e^{-r^2/n}}{i!}$.

As the probability of a match (i.e. two persons choosing the same object) is expected to be larger than 1/2 we set $r^2/n$ to be $\ln 2$. This yields $r \approx 1.117\sqrt{n}$. Since for $a \ll n$, $a\sqrt{n}$ is of the same order of magnitude as $\sqrt{n}$, it means that a match will be found in average after $\mathcal{O}(\sqrt{n})$ persons have chosen an object.

We now investigate how to transform this observation into an effective cryptographic attack.

[1]This approximation will be derived in the homework.

## The birthday attack

Let $h$ be a hash function which digest are of length $n$. The first obvious strategy is to compute $h(x)$ for about $\mathcal{O}\left(\sqrt{n}\right)$ random $x$ and hope for a collision, as the probability is larger than a half.

Example. Let $h$ be a hash function whose output is 128 bits long. Then the above attack leads to a collision in $\mathcal{O}(2^{64})$ steps.

An important drawback in this attack is the amount of storage required, since all the values must be stored in order to be tested for collisions.

Example. What is the hardest: perform $2^{64}$ operations or store $2^{64}$ bytes?

## An improved birthday attack

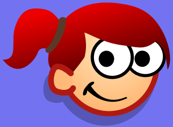
Following the Pollard's rho idea (3.209) it is possible to decrease the amount of storage necessary by computing and comparing two hash sequences.

Select a random initial $x_0$ and then compute $x_i = h(x_{i-1})$ and $x_{2i} = h(h(x_{2(i-1)}))$. At each step compare $x_i$ and $x_{2i}$: a collision on $x_{i-1}$ and $h(x_{2(i-1)})$ is found as soon as $x_i = x_{2i}$.

Note that we implicitly assumed $h$ to act as a random oracle (2.103) such that the probability of having $x_{i-1}$ equal to $h(x_{2(i-1)})$ is very low. In such case no information would be gained.

So far we focused on how to find collisions from a theoretical point of view, that is without considering whether or not the generated hash originates from a meaningful message.

## A real life birthday attack



Alice is happy: she has received a nice contract for a new job in Eve's company.

**Contract**

```
Alice will work 16 hours
a week for a base salary
of 50,000 RMB a month.
Alice can take as much
paid holidays as  she
wants.
```

**Contract**

```
Alice will work 16 hours
a day for a base salary
of 500 RMB a  month;
Alice can take as little
paid holidays as Eve
wants.
```

Eve constructed the two contracts such that they have the same hash. Alice signed one but Eve can pretend it was the other one.

## A real life birthday attack

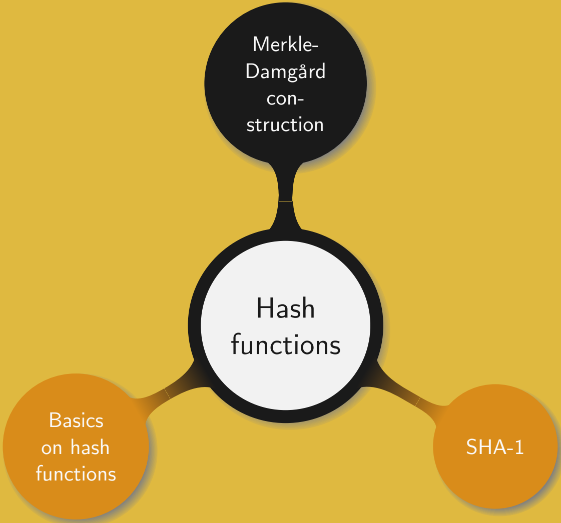
Why is it working?

- Eve generated many good and bad contracts
- She altered each base contract by
  - Changing the punctuation
  - Expressing the same idea using different synonyms
  - Adding extra spaces
- She computed their hash and found a collision

Example. How many ways are there to read this short paragraph?

I {hate|despise|detest} this {terrible|awful|horrible|disastrous} course. It is so {hard|complex|difficult|complicated} and the explanations are always {unclear|confused|doubtful}.

## Chapter organisation



## Designing hash functions

*The goal is to design collision resistant hash functions*

Difficulty:

- Number of possible input: infinite
- Number of possible output: finite

Conclusion: any hash function has an infinite number of collisions

Merkle-Damgård construction: methodology to convert a hash function on strings of fixed length into a hash function accepting arbitrary input lengths

We will prove that if the original hash function is collision resistant then so is the constructed one.

## Iterated hash functions

**Definitions**

1. A function $g$ defined by
$$g : \{0,1\}^{m+t} \longrightarrow \{0,1\}^{m}, \quad t \geq 1,$$
is called a *compression function*.

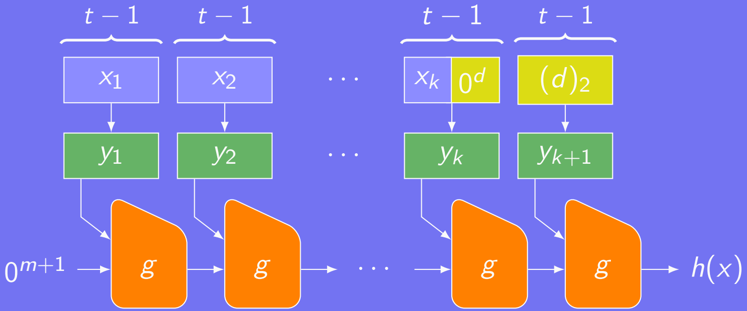
2. A hash function constructed by iteratively applying a compression function is called an *iterated hash function*.

Let $x$ be a bitstring, $|x|$ stands for the length of $x$ and for any bitstring $y$ the concatenation of $x$ and $y$ is denoted $x\|y$. In particular for a bit $x$, $\underbrace{x\|x\|\cdots\|x}_{k \text{ times}}$ is denoted $x^k$.

Let $a$ be an integer, then $(a)_2$ designates its binary representation.

## Merkle-Damgård construction

Let $g : \{0,1\}^{m+t} \to \{0,1\}^m$, $t \geq 2$, be a compression function and $x$ be a bitstring of length $n > 0$.



1. Split $x$ into $k = \lceil \frac{n}{t-1} \rceil$ blocks
2. For $i = 1$ to $k$ set $y_i$ to $x_i$
3. Set $y_k$ to $x_k \| 0^d$
4. Set $y_{k+1}$ to $(d)_2$
5. Compute $z_1 = g(0^{m+1} \| y_1)$
6. For $i = 1$ to $k$
   compute $z_{i+1} = g(z_i \| 1 \| y_{i+1})$
7. Define $h(x)$ as $z_{k+1}$

## Merkle-Damgård theorem

**Theorem** (Merkle-Damgård)

Let $g$ be a collision resistant compression function defined from $\{0,1\}^{m+t}$ into $\{0,1\}^{m}$, with $t \geq 2$. Then the Merkle-Damgård construction is a collision resistant hash function.

Remark. Before proving this theorem we first note that the map $x \mapsto y$ must be injective. In fact if it is not injective then it is possible to find $x \neq x'$ such that $y = y'$. As a result we have $h(x) = h(x')$, that is $h$ is not collision resistant.

## Merkle-Damgård theorem

Proof. Assuming we have a collision on $h$, i.e. $x \neq x'$ and $h(x) = h(x')$, we will prove that a collision on the compression function $g$ can be efficiently found.

First note that if $|x| \neq |x'|$, then they are padded with two different values $d$ and $d'$, respectively. Similarly $k+1$ and $k'+1$ denote the number of blocks for $x$ and $x'$.

Case 1: consider $x \neq x'$ with $|x| \not\equiv |x'| \bmod (t-1)$. Then $d \neq d'$ and $y_{k+1} \neq y'_{k'+1}$. We then have

$$\begin{aligned} g(z_k \| 1 \| y_{k+1}) &= z_{k+1} = h(x) \\ &= h(x') = z'_{k'+1} \\ &= g(z'_{k'} \| 1 \| y'_{k'+1}) \end{aligned}$$

which is a collision on $g$ since $y_{k+1} \neq y'_{k'+1}$.

## Merkle-Damgård theorem

Proof (continued). Case 2a: consider $|x| \equiv |x'| \bmod (t-1)$ with $k = k'$. This implies $y_{k+1} = y_{k'+1}$, and we have

$$\begin{aligned} g(z_k \| 1 \| y_{k+1}) &= z_{k+1} = h(x) \\ &= h(x') = z'_{k+1} \\ &= g(z'_k \| 1 \| y'_{k+1}). \end{aligned}$$

If $z_k \neq z'_k$ then a collision is found. Otherwise we repeat the process and get

$$g(z_{k-1} \| 1 \| y_k) = z_k = z'_k = g(z'_{k-1} \| 1 \| y'_k).$$

Then either we have found a collision or we continue backward until one is obtained. If none is found then we get

$$z_1 = z'_1, \cdots, z_{k+1} = z'_{k+1} \lightning.$$

## Merkle-Damgård theorem

Proof (continued). Case 2b: consider $|x| \equiv |x'| \bmod (t-1)$ with $k \neq k'$. Without loss of generality assume $k' > k$ and proceed as in case 2a. If no collision is found before $k = 1$ then we have

$$\begin{aligned} g(0^{m+1} \| y_1) &= z_1 \\ &= z'_{k'-k+1} \\ &= g(z'_{k'-k} \| 1 \| y'_{k'-k+1}). \end{aligned}$$

By construction the $m+1$st bit on the left is 0 while on the right it is 1. Hence we have found a collision.

All the cases being covered this completes the proof. $\square$

## Constructing hash functions

Remark. We have proven the Merkle-Damgård theorem in the case $t \geq 2$. In fact this is true for any $t \geq 1$ but the case $t = 1$ requires a different approach as $|x| \bmod (t-1)$ cannot be considered[2].

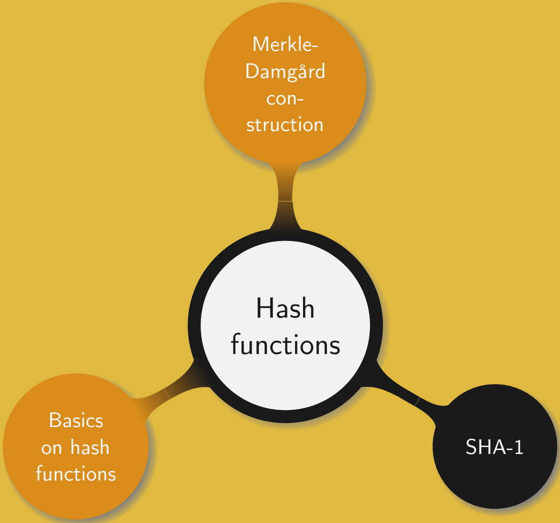
Merkle-Damgård construction served as a basis for the design of various hash functions. Two common examples are MD5 and SHA-1.

MD5 hash function:

- Designed in 1991
- First flaw discovered in 1996
- Collisions found in 2004
- Practical collisions demonstrated in 2005
- Flame malware used collisions on Windows certificates to infect computers... in 2012!

[2] This case will be proven in the homework.

## Chapter organisation



## Secure Hash Algorithm

Secure Hash Algorithm (SHA):

- 1993: SHA-0, 160 bits hash function, never widely adopted
- 1995: SHA-1, similar to SHA-0, with several weaknesses fixed
- 2001: SHA-2, significantly different from SHA-1
- 2005: first attacks against SHA-1
- 2008: SHA-0 totally broken ($< 1$ hour to find collisions)
- 2012: best theoretical attack on SHA-1 in $2^{61}$ operations
- 2012: Keccak hash function is selected to become SHA-3
- 2017: major companies have stopped accepting SHA-1 certificates

## Padding

Given $x$ of length $|x|$:

1. Append 1 to the message
2. Append 0s until the length is $-64 \bmod 512$
3. Append $|x|$ written in base 2 over 64 bits

Let $y$ be the padded value of $x$. By construction $|y| \equiv 0 \bmod 512$. Break $y$ into

$$k = \left\lfloor \frac{|x|}{512} \right\rfloor + 1$$

blocks of 512 bits each.

Example. Assume $|x| = 2800$ bits. Since $2800 \equiv 240 \bmod 512$ append a 1 followed by 207 0s, and the bit representation of 2800 over 64 bits. Thus $y$ is composed of $k = 6$, 512-bit blocks.

## SHA-1 algorithm

As SHA-1 follows the Merkle-Damgård construction it is simply described as an algorithm, while most of the work if performed by the compression function.

Algorithm. (*SHA-1*)

```
Input  : $x$ a bit string
Output: $h(x)$, where $h$ is SHA-1
1 $H_0 \leftarrow 67452301$; $H_1 \leftarrow EFCDAB89$; $H_2 \leftarrow 98BADCFE$;
2 $H_3 \leftarrow 10325476$; $H_4 \leftarrow C3D2E1F0$;
3 $d \leftarrow (447 - |x|) \bmod 512$;
4 $y \leftarrow x \| 1 \| 0^d \| (|x|)_2$;                /* $|x|$ expressed over 64 bits */
5 for $i \leftarrow 1$ to $k$ do
6     $H_0, H_1, H_2, H_3, H_4 \leftarrow$ compress($H_0, H_1, H_2, H_3, H_4, y_i$)
7 end for
8 return $H_0 \| H_1 \| H_2 \| H_3 \| H_4$
```

## SHA-1 compression function

The compression function onto which SHA-1 relies uses:

- The functions $f_0, \dots, f_{79}$ defined by

$$f_i(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{if } 0 \leq i \leq 19 \\ B \oplus C \oplus D & \text{if } 20 \leq i \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \leq i \leq 59 \\ B \oplus C \oplus D & \text{if } 60 \leq i \leq 79 \end{cases}$$

- The constants $K_0, \dots, K_{79}$ defined by

$$K_i = \begin{cases} \text{5A827999} & \text{if } 0 \leq i \leq 19 \\ \text{6ED9EBA1} & \text{if } 20 \leq i \leq 39 \\ \text{8F1BBCDC} & \text{if } 40 \leq i \leq 59 \\ \text{CA62C1D6} & \text{if } 60 \leq i \leq 79 \end{cases}$$

## SHA-1 compression function

**Algorithm.** (*SHA-1 compression function*)

**Input** : Five 32-bit values $H_0, H_1, H_2, H_3, H_4$ and a 512-bit block $y$

**Output** : Five 32-bit values $H_0, H_1, H_2, H_3, H_4$

```
Function compress($H_0, H_1, H_2, H_3, H_4, y$):
    split $y$ into 16 words $W_0, \dots, W_{15}$;
    for $i \leftarrow 16$ to $79$ do
        $W_i \leftarrow ROTL(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})$
    end for
    $A \leftarrow H_0$; $B \leftarrow H_1$; $C \leftarrow H_2$; $D \leftarrow H_3$; $E \leftarrow H_4$;
    for $i \leftarrow 0$ to $79$ do
        $T \leftarrow ROTL^5(A) + f_i(B, C, D) + E + W_i + K_i$;
        $E \leftarrow D$; $D \leftarrow C$;
        $C \leftarrow ROTL^{30}(B)$;
        $B \leftarrow A$; $A \leftarrow T$;
    end for
    $H_0 \leftarrow H_0 + A$; $H_1 \leftarrow H_1 + B$; $H_2 = H_2 + C$; $H_3 = H_3 + D$; $H_4 = H_4 + E$;
    return $H_0, H_1, H_2, H_3, H_4$
end
```

## Comments on SHA-1

Remark.

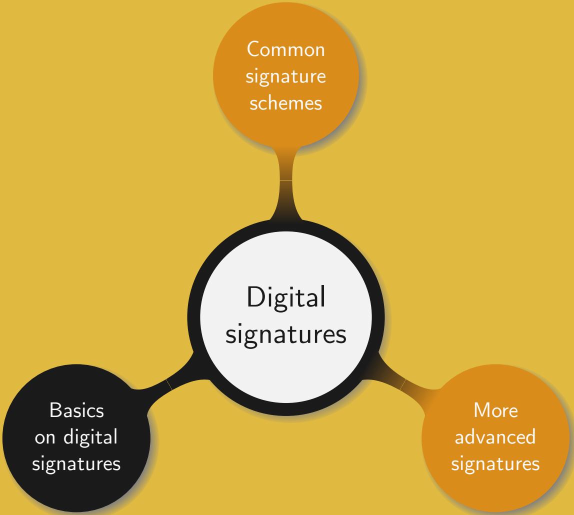
- Compared to SHA-0, SHA-1 only adds *ROTL* in the construction of $W_{16}$ to $W_{79}$
- All the constant in SHA-1 are constructed such as to be above any suspicion
- Weaknesses on SHA-1 lead to the SHA-3 competition
- SHA-2 is a family of six hash functions
- SHA-2 digests can have values 224, 256, 384 and 512
- To date there is no known security issue on any of the SHA-2 hash functions

## Key points

- What is a hash function?
- Why are hash function important in cryptography?
- Explain the birthday paradox
- Describe the Merkle-Damgård construction
- What secure hash function should be used in 2021?

## 5. Digital signatures

## Chapter organisation



## Real life signatures

Middle age:

- Document sealed with a wax imprint of an insignia
- Nobody can reproduce the insignia

Modern time:

- Sign credit card slip
- Compare to the signature at the back of the credit card

Reusing a signature:

- Photocopy
- Cut and paste
- Highly noticeable

## Toward digital signatures

Signing an electronic document:

- Digitalize the signature
- Paste it on the electronic document

Reusing a signature:

- Copy and paste on any document
- Anybody can do it
- Signature is not specific to an individual

Basic idea for a solution:

- Prevent the signature from being separated from its message
- Signature must be easily verified

## Digital signatures

Setup for signatures:

- Message to encrypt is not necessarily secret
- A message might be encrypted after being signed

The signature must be:

- Tied to the signer and to the message being signed
- Easy to verify by anybody
- Hard to forge

*Similar to public key cryptography*

## Attacks on signatures

Similar to attacks on encryption schemes (1.30) we define attacks on signature schemes:

- Key-only attack: Eve has only access to the public key
- Known message attack: Eve has a list of previously signed messages
- Chosen message attack: Eve chooses the messages to be signed
- Selective forgery: Eve is given a message and is able to sign it without being given the private key
- Existential forgery: Eve can find a pair $\langle \mathit{message}, \mathit{signature} \rangle$ without being given the private key

## Signatures and hash functions

Drawback:

- Public key cryptography primitives are used
- Signing a whole message $m$ is then slow

Solution: sign the hash of $m$ using a public hash function

Benefits:

- Faster to generate
- Smaller to store or send
- Conveys the same knowledge as $m$ itself

Given a hash function $h$, denote the signature of the hash of a message $m$ by $\mathrm{sig}(h(m))$

## Signatures and hash functions

Existential forgery:

- Using known message attack:
  1. Get a pair $\langle m, \text{sig}(h(m))\rangle$
  2. Compute $h(m)$ and attempt to find $m'$ such that $h(m) = h(m')$
  3. Considered impossible if $h$ is second pre-image resistant
- Using chosen message attack:
  1. Find two message $m$ and $m'$ such that $h(m) = h(m')$
  2. Persuade the signer to sign $m$
  3. Attach $\text{sig}(h(m)) = \text{sig}(h(m'))$ to $m'$
  4. Considered impossible if $h$ is collision resistant
- Using key-only attack:
  1. Take a signature scheme, without hash function, which is vulnerable to existential forgery using key-only attack
  2. Compute a signature on $h(m)$ for some unknown $m$
  3. Determine such an $m$
  4. Considered impossible if $h$ is pre-image resistant

## Signatures and the birthday attack

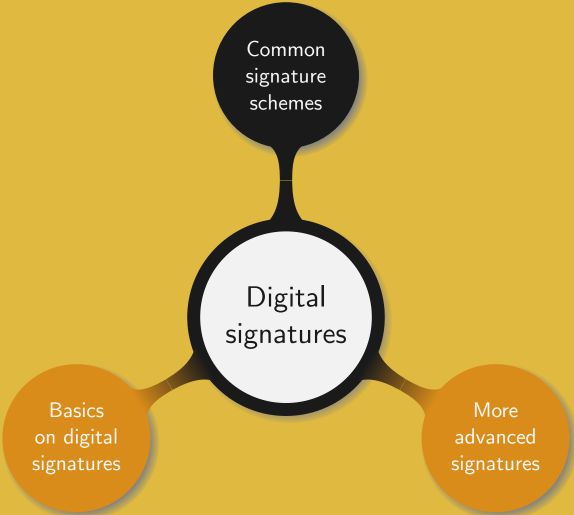
In the previous chapter we investigated the birthday paradox an illustrated how Eve could use this attack to cheat Alice when signing a contract (4.235).

Such an attack can be conducted as soon as the hash is used in place of the whole document. Therefore Alice should be careful and not sign the document. She should rather slightly alter it, for instance by adding a coma or space.

The document being different from the original its hash will be a totally different value. Hence, Eve cannot append Alice signature to the fraudulent contract.

Eve is then defeated and Alice can enjoy a nice contract.

## Chapter organisation



## RSA signatures

Initial setup:

- $p, q$ two primes
- $n = pq$ and $\varphi(n)$
- $e, d$ such that $ed \equiv 1 \bmod \varphi(n)$

- The $n$ from Bob
- The $e$ from Bob

Signature:

- Compute $s \equiv m^d \bmod n$
- Share $m$ and $s$

Verification:

- The message $m$ from Bob
- Compute $m' \equiv s^e \bmod n$
- Compare $m'$ to $m$

## Comments on RSA signatures

Reusing a signature:

- Given a signature $s$ with its message $m$
- Impossible to sign $m'$ using $s$ since $s^e \not\equiv m' \bmod n$

Generating a signature:

- Given a message $m$ find $s$ such that $s^e \equiv m \bmod n$
- This is exactly solving the RSA problem (3.185)

Generating a message:

- Given a signature $s$ generate a message $m \equiv s^e \bmod n$
- It is very unlikely that $m$ is meaningful

## Elgamal signatures

Initial setup:

- $G$ a group of prime order $p$
- $\alpha$ a generator of $G$
- $x$ a secret integer

- $G$ from Bob
- $\alpha$ from Bob
- $\beta \equiv \alpha^x \bmod p$ from Bob

Signature:

- Select a random $k$, with $\gcd(k, p-1) = 1$
- Compute $r \equiv \alpha^k \bmod p$
- Compute $s \equiv k^{-1}(m - xr) \bmod (p-1)$

Verification:

- The triple $\langle m, r, s \rangle$
- Compute
  $$v = \beta^r r^s \equiv \alpha^{xr} \alpha^{k \cdot k^{-1}(m-xr)} \equiv \alpha^m \bmod p$$
- The signature is valid only if $v \equiv \alpha^m \bmod p$

## Elgamal signatures

Example. Set $p = 467$, $\alpha = 2$ and $x = 127$. Then $\beta = 2^{127} \equiv 132 \bmod 467$. The variable $x$ is kept secret, all the others are publicly known.

Signing the message $m = 100$:

- Randomly choose $k = 213$ and keep it since $\gcd(213, 466) = 1$
- Compute $r = 2^{213} \equiv 29 \bmod 467$
- As $k^{-1} \equiv 431 \bmod 466$, $s = 431 \cdot (100 - 127 \cdot 29) \equiv 51 \bmod 466$

To verify the signature $\langle 100, 29, 51 \rangle$, anyone can compute both:

- $132^{29} \cdot 29^{51} \equiv 189 \bmod 467$
- $2^{100} \equiv 189 \bmod 467$

## Comments on Elgamal signatures

First we notice that if $x$ is discovered by an attacker, he can signed any document.
Then we observe that given only a message $m$ he can try to:

- Find $s$ such that
$$\beta^r r^s \equiv \alpha^m \bmod p. \tag{5.1}$$
This can be rewritten $r^s \equiv \beta^{-r}\alpha^m \bmod p$, and finding $s$ means solving the DLP.
- Set $s$ and solve eq. (5.1) for $r$. No feasible solution is known.
- Find $r$ and $s$ simultaneously. It is not known how to do it, but there is no prove that it is impossible to do.

Note that $k$ must remain secret otherwise it is simple to recover $x$. Indeed if $\gcd(r, p-1) = 1$, then $x \equiv (m - ks)r^{-1} \bmod (p-1)$.

## Key-only attack on Elgamal signatures

*Generating a message and its signature only knowing the public key*

Let $i$ and $j$ be two integers such that $0 \leq i, j \leq p-2$. Define $r$ as $\alpha^i\beta^j \bmod p$. Then $\alpha^m$ can be expressed as

$$\alpha^m \equiv \beta^r \left(\alpha^i\beta^j\right)^s \bmod p.$$

Rearranging the different terms yields $\alpha^{m-is} \equiv \beta^{r+js} \bmod p$. This congruence is clearly true if both $m-is$ and $r+js$ are 0 mod $(p-1)$.

Assuming $\gcd(j, p-1) = 1$, we can determine $m$ and $s$ from the two previous equations. Therefore, by construction the signature

$$\langle m, r, s\rangle = \langle -rij^{-1} \bmod (p-1), \alpha^i\beta^j \bmod p, -rj^{-1} \bmod (p-1)\rangle$$

is a valid signature. Note that $m$ is very unlikely to be meaningful.

## Key-only attack on Elgamal signatures

Example. Set $p = 467$, $\alpha = 2$ and $\beta = 132$. Select $i = 99$ and $j = 179$, and then $j^{-1} \equiv 151 \bmod 466$.

The signature is defined by $\langle m, r, s \rangle$ with

$$\begin{cases} r \equiv 2^{99} \cdot 132^{179} & \equiv 117 \bmod 467 \\ s \equiv -117 \cdot 151 & \equiv 41 \bmod 466 \\ m \equiv 99 \cdot 41 & \equiv 331 \bmod 466 \end{cases}$$

The verification is given by

$$132^{117} \cdot 117^{41} \equiv 303 \equiv 2^{331} \bmod 467$$

## Known message attack on Elgamal signatures

Given a valid signature $\langle m, r, s \rangle$ an attacker can construct and sign various other messages.

Generate $h, i$, and $j$ such that $\gcd(hr - js, p - 1) = 1$. Then the triple $\langle m', r', s' \rangle$ defines a valid signature if

$$\begin{cases} r' \equiv r^h \alpha^i \beta^j \bmod p \\ s' \equiv sr'(hr - js)^{-1} \bmod (p-1) \\ m' \equiv r'(hm + is)(hr - js)^{-1} \bmod (p-1) \end{cases}$$

Again this method leads to an existential forgery but cannot be modified into selective forgery. As such those two attacks represent no real threat for Elgamal signatures.

## Misuse of Elgamal signatures

Let $\langle m_1, r_1, s_1\rangle$ and $\langle m_2, r_2, s_2\rangle$ be the two signatures. If they are generated using a common $k$, then $r_1 = r_2 = r = \alpha^k \bmod p$ and

$$\begin{cases} \beta^r r^{s_1} & \equiv \alpha^{m_1} \bmod p \\ \beta^r r^{s_2} & \equiv \alpha^{m_2} \bmod p. \end{cases}$$

Thus $\alpha^{m_1 - m_2} \equiv \alpha^{k(s_1 - s_2)} \bmod p$, and from corollary 3.152 we get

$$m_1 - m_2 \equiv k(s_1 - s_2) \bmod (p-1).$$

Since this congruence has $d = \gcd(s_1 - s_2, p-1)$ solutions (lemma 4.229) it is simple to test all of them and recover $k$. Once $k$ is known $x$ can be recovered as noticed on slide 5.268, and signatures can be forged at will.

## Digital Signature Algorithm

Digital Signature Algorithm (DSA):

- Proposed in 1991 by the NSA
- Adopted as a standard in 1994
- Variant of Elgamal signature scheme
- As in Elgamal the hash of the message is signed
- SHA-1 is the historical choice but SHA-2 (SHA-3) is now recommended
- For a given security level DSA defines two lengths $l_1$ and $l_2$ for the DLP and the hash to feature a balanced security

## Digital Signature Algorithm

### Initial setup:

- A prime $q$, $|q| = l_2$
- A prime $p$, $|p| = l_1$ and $q \mid (p-1)$
- $g$ a generator of $G = \mathrm{U}(\mathbb{F}_p)$
- $\alpha \equiv g^{(p-1)/q} \bmod p$
- $x$ a secret integer

- $p$ from Bob
- $q$ from Bob
- $\alpha$ from Bob
- $\beta \equiv \alpha^x \bmod p$ from Bob

### Signature:

- Select a random $k$, $0 < k < q$
- Compute $r \equiv (\alpha^k \bmod p) \bmod q$
- Compute $s \equiv k^{-1}(m + xr) \bmod q$

### Verification:

- The triple $\langle m, r, s \rangle$
- Compute $v = (\alpha^{s^{-1}m \bmod q}\beta^{s^{-1}r \bmod q} \bmod p) \bmod q$
- The signature is valid only if $v = r$

## DSA signature verification

Observe how the verification works:

By definition of $s$ we know that $m \equiv (-xr + ks) \bmod q$. This implies $s^{-1}m \equiv (-xrs^{-1} + k) \bmod q$. Therefore we can write

$$k \equiv s^{-1}m + xrs^{-1} \bmod q.$$

And we finally get

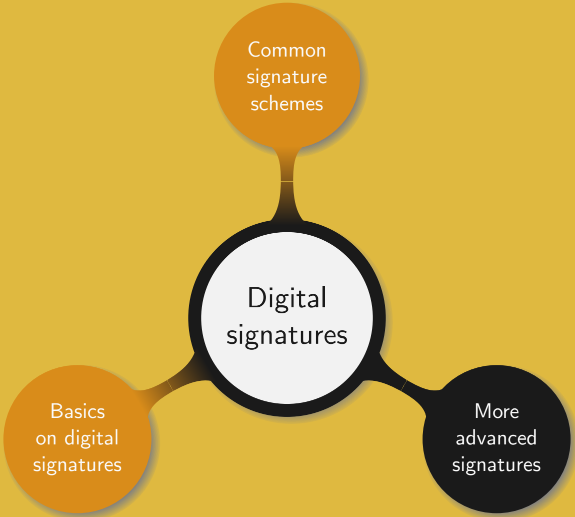
$$\begin{aligned} r &\equiv \alpha^{k} \bmod p \\ &\equiv \alpha^{s^{-1}m + xrs^{-1} \bmod q} \bmod p \\ &\equiv \alpha^{s^{-1}m \bmod q}\beta^{s^{-1}r \bmod q} \bmod p \\ &= v. \end{aligned}$$

## Comments on DSA

Why is DSA different from Elgamal?

- $r$ only "carries part of the information" on $k$
  e.g. if $l_1 = 3072$ and $l_2 = 256$, then about $2^{2816}$ values mod $p$ reduce to a same integer mod $q$
- From the initial setup (slide 5.274), $\alpha^q \equiv 1 \bmod p$. Pohlig-Hellman attack (3.212) does not apply, since $q$ is prime. Not even a little piece of information can be recovered.
- Verification step requires only two modular exponentiations vs. three in Elgamal case

## Chapter organisation



## Blind signatures

Basic idea: sign a document without knowing its content

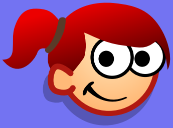
Typical setup: Bob made a new discovery and wants to record it publicly without unveiling it

Strategy: Bob gets his discovery signed by some known authority but without revealing or showing it the content

Danger: what is signed?

## RSA blind signatures

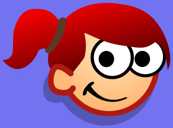
### Initial setup:



- $p, q$ two primes
- $n = pq$ and $\varphi(n)$
- $e, d$ such that $ed \equiv 1 \bmod \varphi(n)$

- $n, e$ from Alice
- A random integer $k$, $\gcd(k, n) = 1$
- For a message $m$ compute $t \equiv k^e m$

### Blind signature:



- $t$ from Bob
- Compute $s \equiv t^d \bmod n$
- Send $s$ to Bob

### Message signature:

- Bob computes
- Bob later shares $m$ and $e$

$$\frac{s}{k} \equiv \frac{t^d}{k} \equiv \frac{k^{ed} m^d}{k} \equiv m^d \bmod n$$

## RSA blind signatures

Remark.

- $k$ being random $k^e \bmod n$ is also random and so is $k^e m \bmod n$
- Alice cannot get any information on what she is signing
- The final value is the same as if Bob had gotten his message signed following the standard procedure
- Verification happens as in “regular RSA signatures”
- There is no need to keep $d$, $p$ and $q$

## Undeniable signatures

Primary goal: design a signature that cannot be verified without the co-operation of the signer

Secondary goals:

- Prevent the signer to disavow a previous signature
- Allow the signer to prove that a forged signature is a forgery

Applications: prevent the illegal distribution of documents without the approval of the author

Structure: composed of three algorithm: signature, verification, and dis-avowal

## Chaum-van Antwerpen signatures

### Initial setup:

- $p$ and $q$ two primes $p = 2q + 1$
- $G$ a subgroup of $\mathbb{F}_p^*$ of order $q$
- $\alpha$ a generator of $G$
- $x$ a secret integer

- $G$ from Bob
- $\alpha$ from Bob
- $\beta \equiv \alpha^x \bmod p$ from Bob

### Signature:

- A message $m$ in $G$
- Compute $s \equiv m^x \bmod p$

### Verification:

1. Choose random $e_1, e_2 \in \mathbb{F}_q^*$
2. $r \equiv s^{e_1}\beta^{e_2} \bmod p$
3. Compute $t \equiv r^{x^{-1} \bmod q} \bmod p$
4. Share $t$ with Alice
5. Valid if and only if $t \equiv m^{e_1}\alpha^{e_2} \bmod p$

## Chaum-van Antwerpen signatures

Remark. On a valid signature we have:

$$t \equiv r^{x^{-1}} \bmod p$$

$$\equiv s^{e_1 x^{-1}} \beta^{e_2 x^{-1}} \bmod p$$

Noting that $s \equiv m^x \bmod p$, and $\beta \equiv \alpha^x \bmod p$, we get

$$t \equiv m^{e_1} \alpha^{e_2} \bmod p.$$

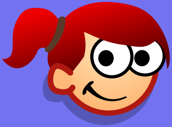
Example. Let $p = 467$, then 2 is a primitive element of $\mathbb{F}_p^*$ and 4 is a generator of the group $G$ of order 233. Taking $x = 101$, $\beta \equiv 4^{101} \equiv 449 \bmod 467$.

Signing the message $m = 119$ yields $119^{101} \equiv 129 \bmod 467$.

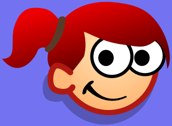
To verify the signature, randomly select $e_1 = 38$ and $e_2 = 397$, then send $r = 13$ while $t = 9$ is replied. Finally test that 9 is congruent to $119^{38} 4^{397} \bmod 467$.

## Chaum-van Antwerpen signatures – Disavowal

2-round verification:



Play the verification protocol using two random values $e_1, e_2 \in \mathbb{F}_q^*$ and expect $t_1 \not\equiv m^{e_1}\alpha^{e_2} \bmod p$



Re-play the verification protocol using two random values $f_1, f_2 \in \mathbb{F}_q^*$ and expect $t_2 \not\equiv m^{f_1}\alpha^{f_2} \bmod p$

Concludes that the signature is a forgery if and only if

$$\left(t_1\alpha^{-e_2}\right)^{f_1} \equiv \left(t_2\alpha^{-f_2}\right)^{e_1} \bmod p$$

## Chaum-van Antwerpen signatures – Disavowal

Remark. The disavowal protocol has two goals:

- Convince Alice that an invalid signature is a forgery
- Prevent Bob from pretending that a valid signature is a forgery

If the signature is invalid then the verification fails. The question is then to know if Bob played a fair game, following the protocol when constructing $t_1$ and $t_2$.

The last step, testing the congruence

$$\left(t_1\alpha^{-e_2}\right)^{f_1} \equiv \left(t_2\alpha^{-f_2}\right)^{e_1} \bmod p,$$

ensures Alice that Bob is not trying to disavow a valid signature.

## From authentication to signature

As investigated earlier (1.80), zero-knowledge proofs can be used to authenticate. In fact this can also be extended to signatures.

General strategy:

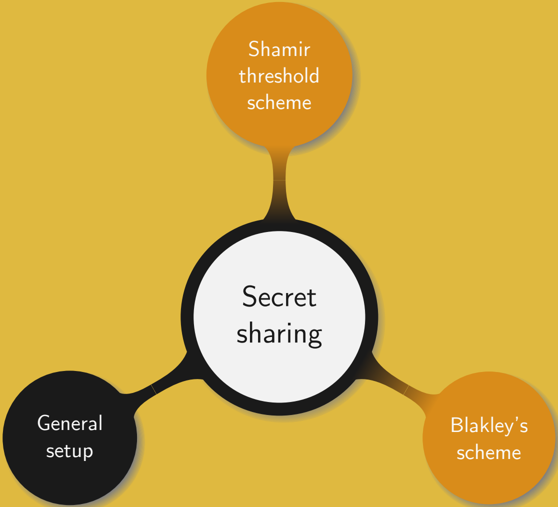
- Send at once all the committed values $C_1, \cdots, C_n$
- For a message $m$ compute $h$, the hash of $\langle C_1, \cdots, C_n, m\rangle$
- Extract $n$ bits, $h_1, \cdots, h_n$, from $h$ to represent the random requests
- Define the signature of $m$ as $\langle h_1, \cdots, h_n, R_1, \cdots, R_n\rangle$, where $R_i$ is the response to challenge $h_i$ for the committed $C_i$
- To ensure a proper security level $n$ should be at least 128

## Key points

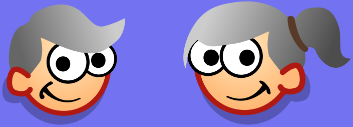
- How to overcome the birthday attack on digital signatures?
- Cite two famous solutions for digital signatures
- What is the reference choice in terms of digital signatures?
- How to transform a zero-knowledge authentication scheme into a signature scheme?

## 6. Secret sharing

## Chapter organisation

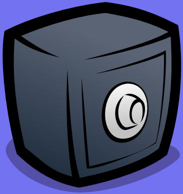


## Intuition



Bob and Alice are now old, famous, and rich

Their mean family wants to get all their money



- All their money is in a safe
- Only them know the secret combination
- They want to teach their family cooperation
- They split the combination such that they need to be at least three to reconstruct it

## Simple secret splitting

Sharing a secret $m$ between two people:

- Generate a random integer $r$
- Give $r$ to a user and $m - r$ to the other

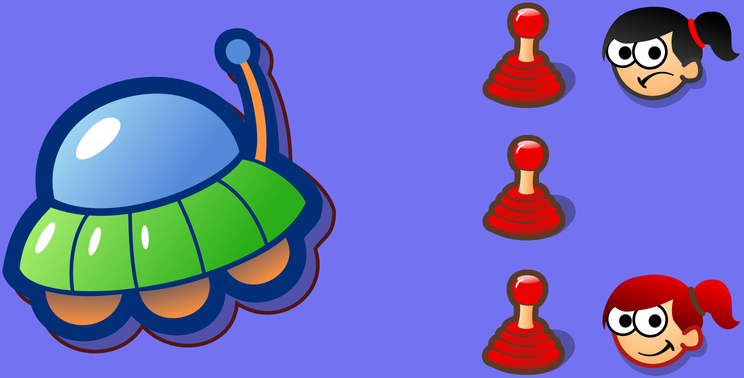
The idea easy extends to the general case of $n$ people:

- Take $l > m$
- Generate $n - 1$ random integers, $r_1, \ldots, r_{n-1}$ between 1 and $l$
- Select $n - 1$ persons and give each an $r_i$, $1 \leq i < n$
- Give the remaining person $m - \sum_{i=1}^{n-1} r_i \bmod l$

## More refined setup

In the previous secret splitting strategy all the participants must be together in order to reconstruct the secret $m$. At times it might be desirable that only a subset of the people is able to reconstruct it.

Example.



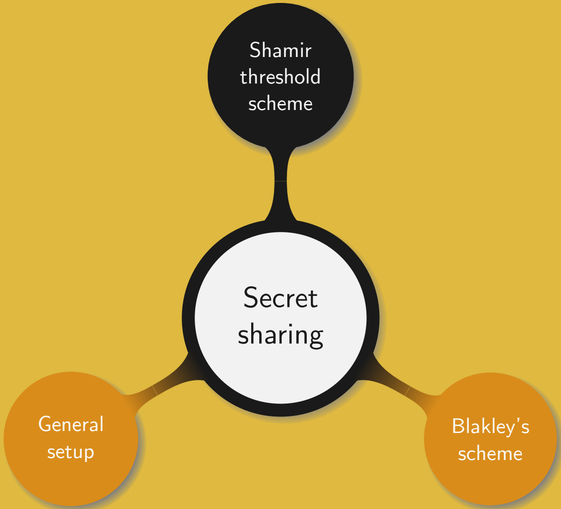
## Secret sharing

**Definition**

Let $t$ and $w$ be two integers such that $t \leq w$. A $(t, w)$-*threshold scheme* is a way to share a secret $m$ among $w$ people, such that any subset of at least $t$ participants can reconstruct $m$, while no smaller subset is able to do it.

In practice, $(t, w)$-threshold schemes constitute a basic building block for many applications where information need to be shared among many users. For instance they can be used for broadcasting.

## Chapter organisation



## Basics on the scheme

Shamir threshold scheme was invented by Shamir in 1979

- Choose a prime $p$ larger than the number of participants and the secret $m$
- Split $m$ among $w$ people such that $t$ persons can reconstruct it
- Choose $t-1$ random integers, $r_1, \dots, r_{t-1}$ mod $p$ and define
$$S(X) = m + r_1 X + \cdots + r_{t-1} X^{t-1} \bmod p$$
- Give each participant a pair $(x_i, y_i)$, with $y_i \equiv S(x_i) \bmod p$
- Keep $S(X)$ secret

*If $t$ people get together and share their pairs they can recover $m$*

## Recovering the secret

Lets see how $t$ people can recover $m$

- Assume the $t$ participants have the pairs $(x_1, y_1), \ldots, (x_t, y_t)$
- They can derive the following expression

$$\underbrace{\begin{pmatrix} 1 & x_1 & \cdots & x_1^{t-1} \\ 1 & x_2 & \cdots & x_2^{t-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & \cdots & x_t^{t-1} \end{pmatrix}}_{V} \begin{pmatrix} m \\ r_1 \\ \vdots \\ r_{t-1} \end{pmatrix} \equiv \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{pmatrix} \bmod p \qquad (6.1)$$

- $V$ is the Vandermonde matrix, which has determinant

$$\det V = \prod_{1 \leq j \leq k \leq t} (x_k - x_j)$$

- Eq. (6.1) has a unique solution when $V$ is invertible
- From theorem 1.53, $V$ is invertible if $\det V \not\equiv 0 \bmod p$, i.e. for all $k$ and $j$, $x_k \not\equiv x_j \bmod p$

## Shamir threshold scheme

Example. We want to construct a (3,8)-threshold scheme to protect the secret message “secret”, which corresponds to $m = 190503180520$.

We choose $p = 1234567890133$ to be larger than $m$ and 8, and generate $r_1 = 482943028839$ and $r_2 = 1206749628665$. Then the polynomial of concern is

$$S(X) = 190503180520 + 482943028839X + 1206749628665X^2.$$

We now distribute the pairs $(x_i, y_i)$, with $1 \leq i \leq 8$:

| $x_i$ | $y_i$ | $x_i$ | $y_i$ |
|---|---|---|---|
| 1 | 645627947891 | 5 | 675193897882 |
| 2 | 1045116192326 | 6 | 852136050573 |
| 3 | 154400023692 | 7 | 973441680328 |
| 4 | 442615222255 | 8 | 1039110787147 |

## Shamir threshold scheme

If 2, 3, and 7 want to recover the message they construct

$$\begin{pmatrix} 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 7 & 49 \end{pmatrix} \begin{pmatrix} m \\ r_1 \\ r_2 \end{pmatrix} \equiv \begin{pmatrix} 1045116192326 \\ 154400023692 \\ 973441680328 \end{pmatrix} \mod 1234567890133.$$

This yields

$$(m, r_1, r_2) = 190503180520, 482943028839, 1206749628665.$$

What if only two participants try to reconstruct $m$?

Remark. A quadratic polynomial is defined by three points, and more generally a polynomial of degree $n$ is defined by $n+1$ points. Therefore if two participants share their information they will still miss a point and as such will not be able to reconstruct the polynomial at discover $m$. Note that there are infinite number of possibilities for this last point.

## More advanced sharing

Example. In a company a secret is split into eight shares. The boss decides eight employees should be required to recover the secret. However he also requests that only four managers or only two board members should be able to recover the secret.

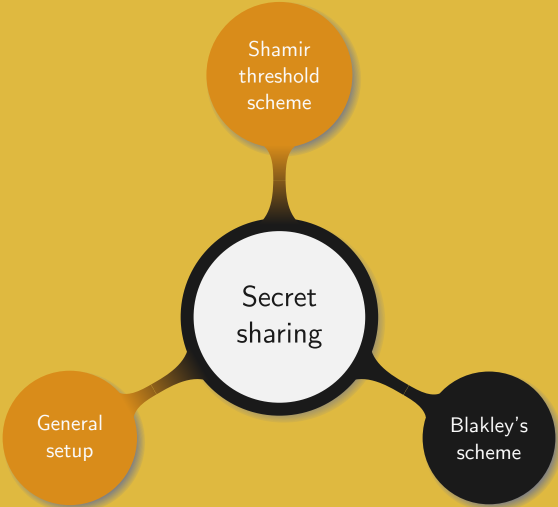
Give each regular employee one share, two to managers, and four to board members. The problem is solved, but note that now one board member together with one manager and two employees can recover the secret.

## More advanced sharing

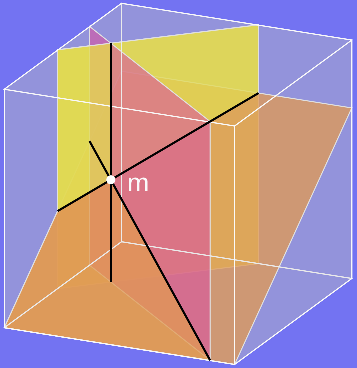
Example. Two companies share a bank vault. They want a setup where four employees from the first company and three from the second are required to be together in order to reconstruct the secret combination.

As each company needs more than 4 or 3 shares, each one could reconstruct the whole secret by itself. The idea is then to write the secret $s = s_1 + s_2$, and give $s_1$ as a shared secret for the first company while $s_2$ becomes a shared secret for the second company. Each of them can apply Shamir threshold scheme to recover its part of the secret. Finally they only need to meet to totally recover the secret combination.

## Chapter organisation



## Basic idea



General rule:
In an *n*-dimensional space, *n* non-parallel hyperplane intersect at a specific point

Cryptographic view:
Give each user the equation of a hyperplane defined such that they all intersect at the secret *m*

## Practical setup

A 3-dimensional setup:

1. Choose a large prime $p$
2. Set $x_s$ to the secret value
3. Select two random values $y_s$ and $z_s$ and define $m = (x_s, y_s, z_s)$
4. Consider the 3-dimensional space mod $p$
5. Give each participant $i$ a plane passing by $m$:
    - Generate two random integers mod $p$, $a_i$ and $b_i$
    - Define $c_i \equiv (z_s - a_i x_s - b_i y_s) \bmod p$
    - The equation of the plane is $z = a_i x + b_i y + c_i$

## Recovering the secret

In a 3-dimensional setup three people can deduce the secret $x_s$:

- Each participant has a plane
$$a_i x + b_i y + c_i \equiv z \bmod p, \quad 1 \leq i \leq 3$$
- They construct the matrix equation
$$\underbrace{\begin{pmatrix} a_1 & b_1 & -1 \\ a_2 & b_2 & -1 \\ a_3 & b_3 & -1 \end{pmatrix}}_{M} \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} \equiv \begin{pmatrix} -c_1 \\ -c_2 \\ -c_3 \end{pmatrix} \bmod p$$
- If $\det M$ is invertible mod $p$ then the system of equations can be solved and $x_s$ can be recovered

## Example

Let $p = 73$, and suppose five people are given the following shares

$$\begin{cases} A: & z = 4x + 19y + 68 \\ B: & z = 52x + 27y + 10 \\ C: & z = 36x + 65y + 18 \\ D: & z = 57x4 - 12y + 16 \\ E: & z = 34x + 19y + 49 \end{cases}$$

$A$, $B$, and $C$ decide to recover the secret:

$$\begin{pmatrix} 4 & 19 & -1 \\ 52 & 27 & -1 \\ 36 & 65 & -1 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} \equiv \begin{pmatrix} -68 \\ -10 \\ -18 \end{pmatrix} \bmod 73$$

The solution yields $x_s = 42$, $y_s = 29$, and $z_s = 57$

## Blakley vs. Shamir

### Blakley's scheme

- Matrix $M$ not always invertible
- Hard to select $a_i$, $b_i$, and $c_i$ for $M$ to be always invertible
- More general setup
- Much information carried by each participant $(a_i,\ b_i, \cdots)$

### Shamir's threshold scheme

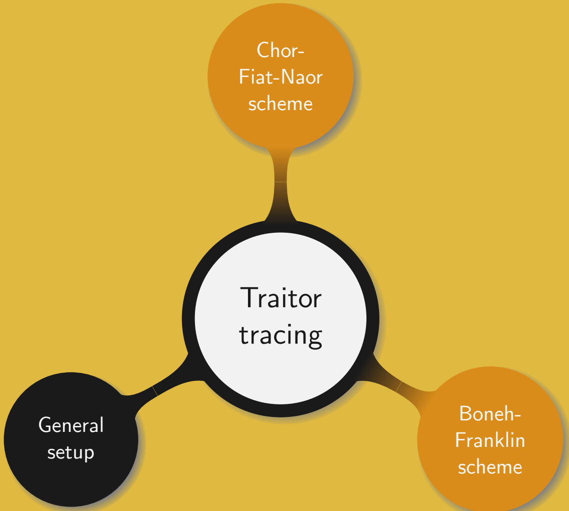
- Matrix $V$ is invertible, as long as no two shares are congruent mod $p$
- Method can be view as a particular case of Blakley
- Little information carried by each participant $(x_i, y_i)$

## Key points

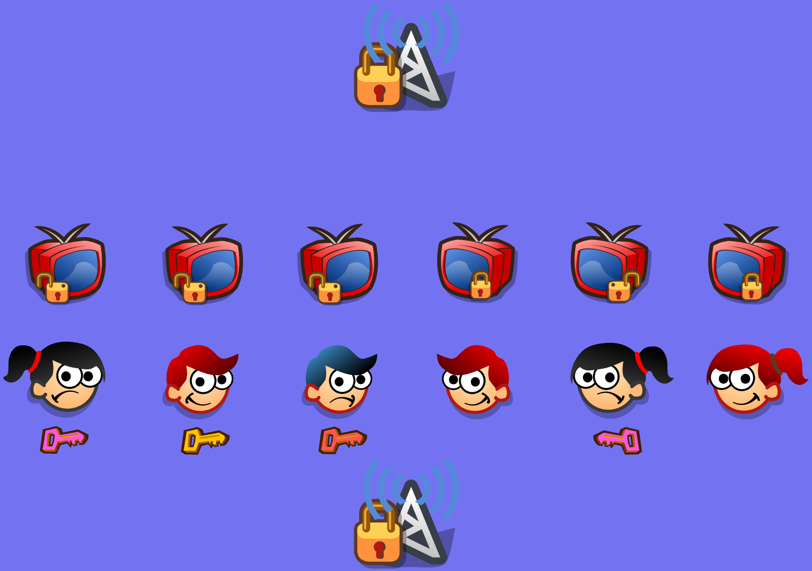
- Explain what is secret sharing
- Describe Shamir's threshold scheme
- What is the key idea behind Blakley's scheme?
- Provide several examples where secret sharing is useful

## 7. Traitor tracing

## Chapter organisation



## A basic PayTV setup



## A (too) simple solution

In this setup each user:

- Generates a pair of public/private keys
- Shares his public key with the service provider
- Receives the broadcast encrypted using his public key
- Decrypts and enjoys the program

Drawback: inefficient due to the amount of bandwidth required

Improving the solution: design a scheme where the encrypted information can be decrypted using different secret keys

## Possible attacks

Three general types of attack:

- Decrypt the broadcast and share it
- Record the encrypted broadcast and share the decryption key with other people such that they can watch it
- Create a new secret key from several secret keys collected from various users

The two first cases are not traceable. The third scenario allows the construction of pirate decoders which can be sold at a large scale. The goal is to construct a scheme where tracing the “traitors” who shared their secret key is possible.

## Tracing goals

Desirable properties of the scheme:

- Allows to trace the piracy
- Prevent legitimate users from being incriminated or framed by the traitors
- Allows the disconnection of illegitimate users
- Supplies legal evidence of the pirate's identity

## Generalities

Types of schemes:

- Symmetric vs. asymmetric: how is encryption done
- Static vs. dynamic: keys changes at certain intervals
- Alternate approach: include credit card number in the user's key or use watermarking

Components of a Traitor Tracing scheme:

- Key generation and distribution
- Encryption and decryption methods
- Tracing algorithm

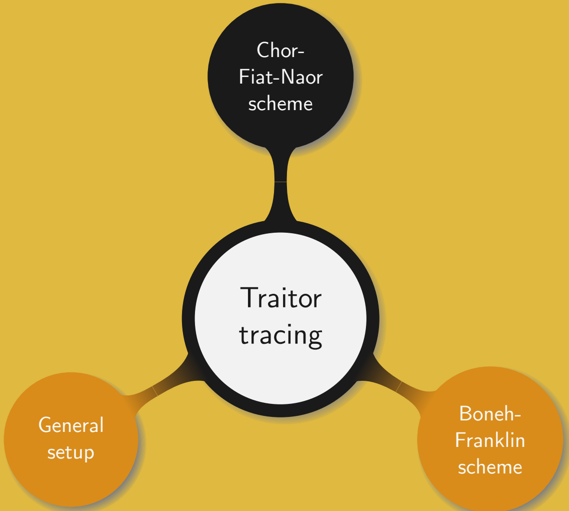
## Resilience

When several users collude to generate a new pirate decoder, they use some personal information. The goal of the tracing algorithm is then to spot at least one traitor.

**Definition**

Assuming the underlying encryption to be secure, a scheme is said to be *$m$-resilient* if it can trace at least one traitor from a coalition of at most $m$ malicious users.

## Chapter organisation



## Method

Given $n$ users $u_1, \dots, u_n$ and $2\log n$ keys

$$k_{1,0}, k_{1,1}, k_{2,0}, \dots, k_{\log n,0}, k_{\log n,1},$$

define the key $K_i$ of user $u_i$ by

$$K_i = \langle k_{1,b_{i,1}}, k_{2,b_{i,2}}, \dots, k_{\log n, b_{i,\log n}} \rangle,$$

where $b_{i,j}$ is the $j$-th bit in the binary representation of $i$.

Applying this strategy, minimizes the number of keys as well as the bandwidth necessary to transmit the encrypted program to all the users.
Example. For eight users six keys $k_{1,0}, k_{1,1}, k_{2,0}, \dots, k_{3,1}$ are defined. Since $(5)_{10} = (101)_2$, user $u_5$ has key $K_5 = \langle k_{1,1}, k_{2,0}, k_{3,1} \rangle$.

## Method

Given some information $m$ to broadcast, it is encrypted using a symmetric encryption protocol $E$ with a secret key $S$. Then proceed as follows.

- Choose $s_i$, $1 \leq i \leq \log n$ such that

$$S = s_1 \oplus s_2 \oplus \cdots \oplus s_{\log n}$$

- Encrypt $s_i$ using $E$ and $k_{i,0}, k_{i,1}$
- Broadcast both the encrypted version of $m$ and of the secret key $S$.

As each user $u_i$, $1 \leq i \leq n$, knows either $k_{i,0}$ or $k_{i,1}$, everybody can recover the secret key $S$ and then decrypt the information $m$ contained in $E_S(m)$.

## Formalism

**Definitions**

Let $E$ be a symmetric encryption protocol with keys of size $l$.

1. A *codeword* is a $k$-tuple of elements from $\mathbb{F}_q$, where $q = 2^l$
2. A set of codewords is called a *code*
3. Let $\mathcal{C} \subset \left(\mathbb{F}_q\right)^k$ be a code and $d = \langle d_1, \dots, d_k \rangle$ be a codeword that is not in $\mathcal{C}$. If for all $1 \leq i \leq k$ there exists a codeword $c = \langle c_1, \dots, c_k \rangle$ in $\mathcal{C}$ such that $d_i = c_i$, then $d$ is called a *descendant* of $\mathcal{C}$. All the descendants of $\mathcal{C}$ form a *descendant code* of $\mathcal{C}$, denoted $\mathrm{desc}(\mathcal{C})$
4. Let $d$ be a descendant of $\mathcal{C}$ and $\mathcal{S}_d = \left\{\mathcal{C}_p \subseteq \mathcal{C} : \ d \in \mathrm{desc}(\mathcal{C}_p)\right\}$. A codeword $c \in \mathcal{C}$ is an *identifiable parent* for $d$ if
$$c \in \bigcap_{\mathcal{C}_p \in \mathcal{S}_d}$$

## Explanations

In the context of a PayTV the previous definitions can be interpreted by identifying each codeword to a decoder.

The idea is then to define a code $\mathcal{C}$ by assigning a codeword to each decoder, in such a way that $\mathrm{desc}(\mathcal{C}) \bigcap \mathcal{C}$ is empty.

The key used in a pirate decoder being constructed from elements of $\mathcal{C}$, it is a descendant of $\mathcal{C}$. Then $\mathcal{S}_d$ defines the set of suspects who could be involved in the generation of $d$.

An *identifiable parent* $c$ from $\mathcal{S}_d$ is a suspect decoder which can be identified as guilty, since $d$ is derived from $c$.

## Explanations

Example. Let $\mathcal{C}$ be the code defined by

$$
\begin{array}{llll}
c_1 = \langle 0,0,0\rangle, & c_2 = \langle 0,1,1\rangle, & c_3 = \langle 0,2,2\rangle, & c_4 = \langle 1,0,3\rangle, \\
c_5 = \langle 2,0,4\rangle, & c_6 = \langle 3,3,0\rangle, & c_7 = \langle 4,4,0\rangle. &
\end{array}
$$

Assume that among the $c_i$, $1 \leq i \leq 7$, two traitors collude to construct a codeword $d = \langle d_1, d_2, d_3\rangle$. If any coordinate of $d$ is non-zero then at least one parent can be identified:

$$
\begin{array}{llll}
d_1 = 1 \rightarrow c_4, & d_1 = 2 \rightarrow c_5, & d_1 = 3 \rightarrow c_6, & d_1 = 4 \rightarrow c_7, \\
d_2 = 1 \rightarrow c_2, & d_2 = 2 \rightarrow c_3, & d_2 = 3 \rightarrow c_6, & d_1 = 4 \rightarrow c_7, \\
d_3 = 1 \rightarrow c_2, & d_3 = 2 \rightarrow c_3, & d_3 = 3 \rightarrow c_4, & d_3 = 4 \rightarrow c_5.
\end{array}
$$

Finally if $d = \langle 0,0,0\rangle$, then $c_1$ is an identifiable parent.

## Distance

**Definitions**

1. The *hamming distance* between two elements $a$ and $b$ of $\left(\mathbb{F}_q\right)^k$ is defined as $\operatorname{dist}(a, b) = |\{i : a_i \neq b_i, 1 \leq i \leq k\}|$
2. Let $\mathcal{C}$ be a code, then the minimal distance of $\mathcal{C}$ is

$$\operatorname{dist}(\mathcal{C}) = \min\left\{\operatorname{dist}(a, b) : a, b \in \mathcal{C}, a \neq b\right\}$$

Example. Reusing the code from example 7.321 we note that $\operatorname{dist}(c_1, c_i)$, $2 \leq i \leq 7$, is 2, while $\operatorname{dist}(c_2, c_4) = 3$. We can observe that no distance is smaller than 2 such that $\operatorname{dist}(\mathcal{C}) = 2$.

## Attributing the codewords

We now introduce a result which provides some hint on how to choose the distance in order to be able to identify at least one parent of an illegal decoder.

**Theorem**

Let $\mathcal{C} \subset (\mathbb{F}_q)^k$ be a code of length $k$ and minimal distance $D$. If $D > k(1 - 1/w^2)$, where $w$ is the size of the coalition, then it is possible to identify a parent of a descendant of $\mathcal{C}$.

Proof. For any $a, b$ in $(\mathbb{F}_q)^k$, we define $\mathrm{match}(a, b) = k - \mathrm{dist}(a, b)$. Let $\mathcal{S}_d = \{\mathcal{C}_p \subseteq \mathcal{C} : d \in \mathrm{desc}(\mathcal{C}_p)\}$ denote the set of suspects and $d$ be a descendant of $\mathcal{C}_p \subset \mathcal{S}_d$. Let $c$ be the closest element from $d$. We will now prove that $c$ belongs to $\mathcal{C}_p$.

## Attributing the codewords

Proof (continued). First note that since $d$ is a descendant of $\mathcal{C}_p$, it follows that

$$\sum_{c' \in \mathcal{C}_p} \operatorname{match}(d, c') \geq k.$$

Then as the coalition features $w$ users it means that $|\mathcal{C}_p| \leq w$, and we can find a codeword $c'$ in $\mathcal{C}_p$ such that

$$\operatorname{match}(d, c') \geq \frac{k}{w}.$$

Recalling that $c$ is the closest element from $d$ we get

$$\operatorname{match}(d, c) \geq \frac{k}{w}.$$

## Attributing the codewords

Proof (continued). Finally we consider the number of common coordinates between $b \in \mathcal{C} \backslash \mathcal{C}_p$ and $d \in \text{desc}(\mathcal{C}_p)$

$$\begin{aligned} \text{match}(d, b) &\leq \sum_{c' \in \mathcal{C}_p} \text{match}(c', b) \\ &\leq w(k - D). \end{aligned}$$

If $D > k(1 - 1/w^2)$, then clearly $\text{match}(d, b) < \text{match}(d, c)$. Since this is true for any $b \notin \mathcal{C}_p$, this means that $c$ belongs to $\mathcal{C}_p$. □

This result is extremely useful as it provides information on how to construct the code and appropriately select the distance in order to trace traitors. As a general rule, the larger the minimum distance between two codewords, the easier to trace. On the other hand, having a large minimum distance will decrease the number of possible codewords in the code.

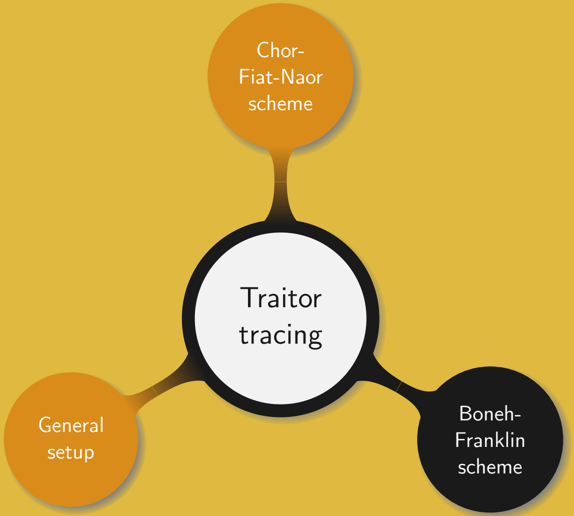
## Back to Chor-Fiat-Nahor scheme

We notice the following properties:

- The scheme is using symmetric cryptography
- The number of decoders is $n$
- Each decoder is represented by a $k$-tuple of $\mathbb{F}_q$, with $k = \log n$
- The scheme is 1-resilient

*The key aspect of this method is to choose a "good code"*

## Chapter organisation



## The representation problem

**Problem** (Representation Problem)

Let $G$ be a cyclic group of order $n$ and $g_1, \cdots, g_m$, be $m$ distinct generators of $G$. Then any element $y \in G$ can be expressed as $\prod_{i=1}^{m} g_i^{e_i}$, for some $0 \leq e_i \leq \varphi(n)$. We say that $(e_1, \cdots, e_m)$ is a representation of $y$ in the base $(g_1, \cdots, g_m)$. Given $G$, $y$ and a base $(g_1, \cdots, g_m)$, find the representation of $y$.

This problem can be seen as a generalisation of the DLP (3.205). Moreover when the generators are chosen randomly, finding two different representations of a given element is as hard as solving the DLP.

## Setup

Simple description:

- $p$ is prime
- $G$ is a subgroup of prime order $q$
- $g$ is a generator of $G$
- $m$ is the maximal size of the coalition the scheme can trace
- $l \geq 2m + 2$ is the number of private keys
- $\mathcal{C} = \{c_1, \cdots, c_l\}$ is a code of $\mathbb{Z}^{2m}$

The scheme now described is CPA-1 secure but it can be extended into an enhanced CCA-2 version. This has the effect of more closely mirroring a real life context.

## Key generation

The public and private keys are generated as follows:

1. Choose $2m$ random elements $r_i$, $1 \leq i \leq 2m$, in $\mathbb{F}_q$ and for each $r_i$ compute $g_i = g^{r_i}$
2. Set the public key to $\langle y, g_1, \cdots, g_{2m} \rangle$, where $y = \prod_{i=1}^{2m} g_i^{\alpha_i}$, with the $\alpha_i$ being random elements from $\mathbb{F}_q$
3. Set the private key $k_i \in \mathbb{F}_q$ such that $k_i c_i$ is a representation of $y$ in the base $(g_1, \cdots, g_{2m})$. That is

$$k_i = \frac{\sum_{j=1}^{2m} r_j \alpha_{i_j}}{\sum_{j=1}^{2m} r_j c_{i_j}} \bmod q$$

## Encryption and decryption

Encryption:

- A message $M$ in $G$
- Generate a random $a$ in $\mathbb{F}_q$
- Define the ciphertext as $C = \langle My^a, g_1^a, \cdots, g_{2m}^a \rangle$

Decryption:

- A ciphertext $C = \langle My^a, g_1^a, \cdots, g_{2m}^a \rangle$
- Use the $i$-th secret key $k_i$ to compute $U = \left( \prod_{j=1}^{2m} \left( g_j^a \right)^{c_{i_j}} \right)^{k_i}$

$$\begin{aligned} U &= \left( g^{\sum_{j=1}^{2m} r_j c_{i_j}} \right)^{k_i a} \\ &= \left( g^{\sum_{j=1}^{2m} r_j \alpha_{i_j}} \right)^{a} \\ &= y^a \end{aligned}$$

- Recover $My^a / U = M$

## Tracing

The tracing algorithm being more advanced we do not detail it here but only highlight the main ideas.

The key principle behind the tracing ability is related to the difficulty of finding new representations. In fact if several users collude they are able to construct a new representation $y$. However this construction leads to a so called “convex combination” of the traitor’s keys. It can be proved that if one can find a new representation that is not a convex combination of already known representations then one can solve the DLP.

By analysing the newly generated representation it is then possible to trace at least one traitor.

## Revocation

Traitor tracing in practice:

- Simple case: the value of the constructed $k$ is known
- Harder case:
  - Only a card containing the key $k$ is available
  - Not possible to directly read $k$
  - Blackbox traitor tracing

Key revocation:

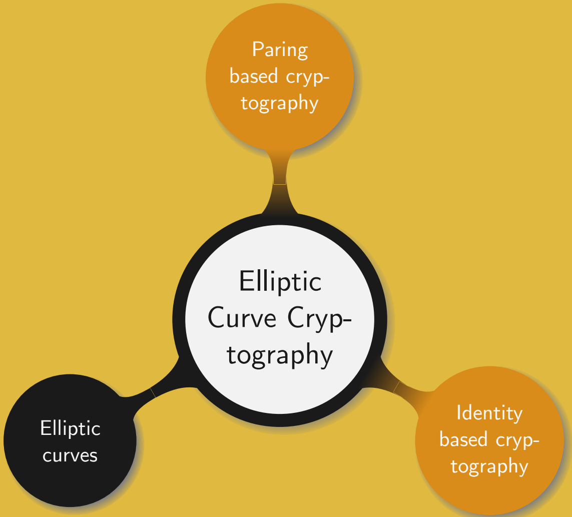
- Black list: too complex
- Alternative:
  - Define the set of all the secret keys $\mathcal{K} = \{k_1, \cdots, k_m\}$
  - For each subset $\mathfrak{K}$ of $\mathcal{K}$ generate a public key whose encryption can only be decrypted by the keys in $\mathfrak{K}$
  - To revoke a secret key change the encryption key

## Key points

- List the type of attacks a multicast system can undergo
- What is the resilience?
- Explain the basic idea to trace traitors
- How to handle key revocation?

## 8. Elliptic Curve Cryptography

## Chapter organisation


Paring based cryp-tography
Elliptic Curve Cryp-tography
Elliptic curves
Identity based cryp-tography

## Complexity and security

Main public key cryptography problems:

- RSA problem (3.185)
- Discrete Logarithm Problem (3.205)

Both problems can be solved using algorithms with sub-exponential complexity; that is algorithm with complexity neither polynomial nor exponential but somewhere in-between.

Consequence on the key size:

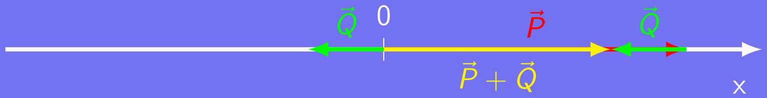
| Security level (bits) | 80 | 112 | 128 | 192 | 256 |
|---|---|---|---|---|---|
| Key size (bits) | 1024 | 2048 | 3072 | 7680 | 15360 |

## Toward Algebra and Geometry

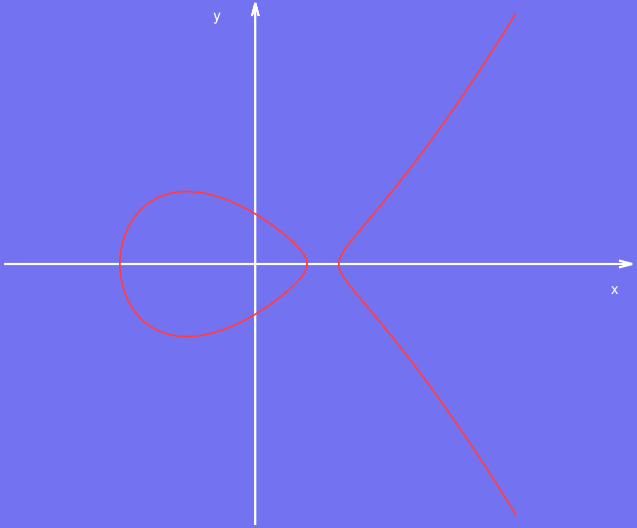
In chapter 3 it was noted that Pollard's rho was a generic algorithm solving the DLP (remark 3.209). In contrast with more efficient algorithms, such as the NFS, Pollard's rho algorithm does not take advantage of the underlying structure of the group.

Therefore a simple idea for the DLP consists in finding a group where no algorithm performs better than Pollard-rho (3.207).

Abstract algebraic structures can be be studied from the perspective of geometry. A simple example is the group structure of the integers $(\mathbb{Z}, +)$ which can be represented on the number line.



## A new shape



## Elliptic curves

The red curve:

- Is called an *elliptic curve*
- Is defined over a field, here the reals
- Can be defined over other fields
- Is given by the equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{8.1}$$

In most cases, a change of variable allows to rewrite equation (8.1) in the more simple form

$$y^2 = x^3 + bx + c$$

## An elliptic curve as a group

By construction this is almost a group: only a unit element is missing. Therefore we adjoin the point $\mathcal{O}$, called *point at the infinity*. This point can be viewed as the point where all the vertical lines intersect.

**Proposition**

Let $E$ be an elliptic curve of equation $y^2 = x^3 + bx + c$. Taking two point $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on $E$, we define the addition law over $E$ by $P_1 + P_2 = P_3 = (x_3, y_3)$ by

$$x_3 = m^2 - x_1 - x_2, \qquad y_3 = m(x_1 - x_3) - y_1,$$

with

$$m = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } P_1 \neq P_2 \\ (3x_1^2 + b)/(2y_1) & \text{if } P_1 = P_2. \end{cases}$$

This addition law is both associative and commutative. If taking $\mathcal{O}$ as unit element, then $E$ is an abelian group.

## Elliptic curves modulo $p$

Example. Let $E$ be the elliptic curve defined by $y^2 \equiv x^3 + 4x + 4 \bmod 5$. The points on $E$ are all the pairs of elements $(x, y)$ in $\mathbb{F}_5 \times \mathbb{F}_5$ that satisfy the equation.

| $x$ mod 5 | $y^2$ mod 5 | $y$ mod 5 | Points on $E$ |
|---|---|---|---|
| 0 | 4 | 2 or 3 | (0,2) and (0,3) |
| 1 | 4 | 2 or 3 | (1,2) and (1,3) |
| 2 | 0 | 0 | (2,0) |
| 3 | 3 | | |
| 4 | 4 | 2 or 3 | (4,2) and (4,3) |

The elliptic curve $E$ has eight points: seven calculated from the equation plus the point at the infinity $\mathcal{O}$.

## Elliptic curve modulo $p$

Example. We now determine the sum of the two points (1,2) and (4,3) on $E$.

First we note that as 3 is invertible mod 5 then

$$m = \frac{3-2}{4-1} \equiv 2 \bmod 5.$$

Then we compute $x_3$ and $y_3$,

$$\begin{aligned} x_3 &\equiv 2^2 - 1 - 4 \equiv 4 \bmod 5 \\ y_3 &\equiv 2(1-4) - 2 \equiv 2 \bmod 5. \end{aligned}$$

Finally we have $(1,2) + (4,3) = (4,2)$ on $E$.

## Number of points modulo $p$

Simple strategy to count points on any elliptic curve mod $p$:

- Compute $t = x^3 + bx + c$ for $0 \leq x \leq p - 1$
- If $t$ is square then $(x, \sqrt{t})$ and $(x, -\sqrt{t})$ are on $E$
- Approximately one over two values of $t$ are squares
- An elliptic curve mod $p$ has about $p$ points

**Theorem** (Hasse's theorem)

If $E$ is an elliptic curve with $n$ points, then

$$|n - p - 1| < 2\sqrt{p}.$$

## Elliptic Curve Discrete Logarithm Problem

Since an elliptic curve gives rise to a group structure it means that we can define a discrete logarithm problem on them.

**Problem** (Elliptic Curve Discrete Logarithm Problem (ECDLP))

Let $E$ be an elliptic curve over a finite field $\mathbb{F}_q$, $q = p^n$ for some prime $p$ and integer $n$, and $P$ be a generator of the group. Given a point $Q$ on the $E$, find $k$ in $\mathbb{N}$ such that $[k]P = Q$, where $[k]P$ represent the operation of adding $k - 1$ times the point $P$ to itself.

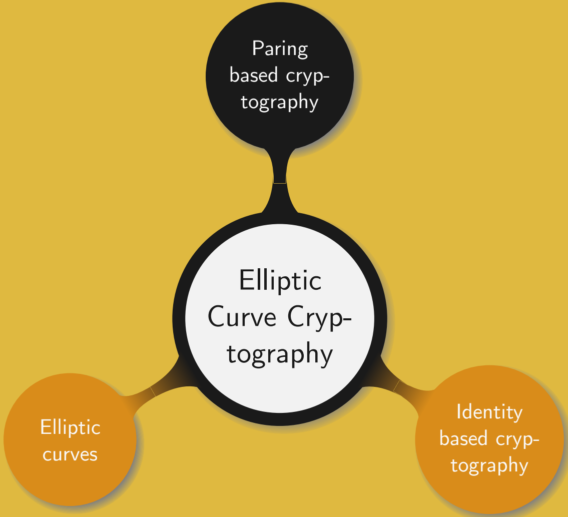
From a geometrical point of view it is clear that given $k$ and $P$ is it easy to find $[k]P$. However given $Q$ and $P$ is it hard to determine $k$ such that $[k]P = Q$. Therefore the ECDLP allows the definition of a 1-way function.

## Security of the ECDLP

In the general case, the best known algorithm to solve the ECDLP is Pollard's rho algorithm. From a cryptographic angle it means that the key size, in terms of bits, is only twice the security level.

| Security level (bits) | Key size (bits) | |
|---|---|---|
| | DLP | ECDLP |
| 80 | 1024 | 160 |
| 112 | 2048 | 224 |
| 128 | 3072 | 256 |
| 192 | 7680 | 384 |
| 256 | 15360 | 512 |

## Chapter organisation


Paring based cryp-tography
Elliptic Curve Cryp-tography
Elliptic curves
Identity based cryp-tography

## Solving the ECDLP

In remark 3.213 we noted that it was possible to map a hard instance of the DLP into an easier one, for instance in an additive group. Unfortunately this strategy is not practical since computing the map is too time consuming and as such would not provide an speedup.

The question now needs to be reconsidered in the case of elliptic curves. As mentioned earlier (8.346) the best algorithm to solve the ECDLP has exponential complexity. Therefore exhibiting a map from an elliptic curve into a subgroup of a finite field could bring much improvement in solving the ECDLP.

Although such maps exist only a few families of elliptic curves are vulnerable to this attack as in most cases the map is again to hard to compute. In the case where is can be efficiently computed it is called a *cryptographic pairing*.

## Pairing

**Definition**

A *cryptographic paring* is a map $e$ from two additive groups $G_1$ and $G_2$ into a multiplicative group $G_T$. For some given $P_1$, $P_2$, $P \in G_1$ and $Q_1$, $Q_2$, $Q \in G_2$ a pairing has the following properties:

- *Bilinearity:*

$$\begin{aligned} e(P, Q_1 + Q_2) = & \quad e(P, Q_1)e(P, Q_2) \\ e(P_1 + P_2, Q) = & \quad e(P_1, Q)e(P_2, Q), \end{aligned}$$

- *Non-degeneracy:*

$$\begin{aligned} &\forall P \in G_1,\ P \neq \mathcal{O} \quad \exists Q \in G_2 \text{ such that } e(P, Q) \neq 1 \\ &\forall Q \in G_2,\ Q \neq \mathcal{O} \quad \exists P \in G_1 \text{ such that } e(P, Q) \neq 1, \end{aligned}$$

- The map $e$ is efficiently computable.

## Pairings in practice

History of elliptic curves in cryptography:

- Discovered in the mid 80es
- In the 90es pairings were used to attack the ECDLP
- Then some families were abandoned since they were insecure
- Around 2000 pairings were used in a "constructive way"

The most useful property of a pairing is bilinearity. It was realised that it could be used to construct new efficient protocols. We now describe one such example, due to Joux, where three parties can construct a common secret key in only one round.

Notations:

- For $p$ a prime and $n$ an integer, $q = p^n$
- An elliptic curve over $\mathbb{F}_q$, $E(\mathbb{F}_q)$
- A subgroup of $E(\mathbb{F}_q)$, $G = G_1 = G_2$

## Tripartite key exchange protocol

Initial setup:

Common: $G$ a subgroup of $E(\mathbb{F}_q)$, and $P$ a generator of $G$
Personal: a secret key $x_b$ (Bob), $x_a$ (Alice), or $x_c$ (Charly)

Key broadcasting:

Common: broadcast $Q_i = [x_i]P$, $a \leq i \leq c$
Personal: $e(Q_a, Q_c)^{x_b}$ (Bob), $e(Q_b, Q_c)^{x_a}$ (Alice), or $e(Q_a, Q_b)^{x_c}$ (Charly)
Shared secret key:

Common: $e([x_a x_b x_c]P, P) = e(P, P)^{x_a x_b x_c}$

## Security of pairings

Security:

- $x_a$, $x_b$, and $x_c$ must remain secret
- $e(P, P)^{x_a x_b x_c}$ must remain secret

Conclusion: both the DLP and the ECDLP must be secure

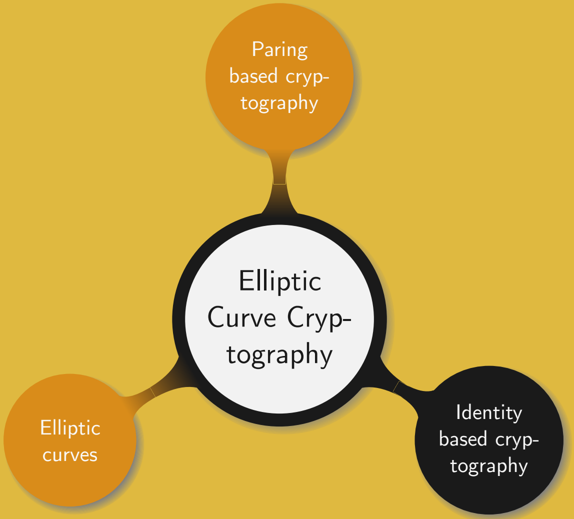
Efficiency:

- Pairings become more expensive as $E(\mathbb{F}_q)$ gets larger
- The group $G_T$ is a subgroup of $\mathbb{F}_{q^k}$, for some integer $k$
- Arithmetic in $\mathbb{F}_{q^k}$ becomes more expensive as $p$, $n$, and $k$ grow

Conclusion: balance both security and efficiency

*Always ensure both the ECDLP and DLP have a similar security level*

## Chapter organisation


Paring based cryp-tography
Elliptic Curve Cryp-tography
Elliptic curves
Identity based cryp-tography

## From PKC to IBC

All the protocols presented in chapter 3 require the use of a directory. This implies first that the user must register with a directory provider when he generates his keys but also that any other user who wants to communicate with should must connect to the directory in order to retrieve a public key.

An alternative, and more convenient solution, would be to use the identity of a user to automatically generate his public key. This would in turn eliminate the necessity of a directory.

Two common ways to solve this problem are to use pairings, or lattice based cryptography. We now present the first identity based protocol proposed by Boneh and Franklin in 2001.

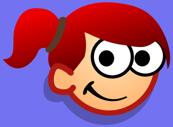
## Initial setup

Trusted Authority (TA)

The TA prepares the system:

- Select an elliptic curve $E(\mathbb{F}_q)$
- Choose $G$, a subgroup of $E(\mathbb{F}_q)$, and $P$ a generator of $G$
- Pick a random $s$ and set $Q = [s]P$
- Choose a hash function $H_1$ mapping a string into a point in $G$
- Choose a hash function $H_2$
- For each user identity $ID$ compute the secret key
$$s_{ID} = [s]H_1(ID)$$
- Public parameters: $\langle H_1, H_2, G, G_T, P, Q \rangle$

## Encryption



Alice sends a message to Bob

Given a message $m$:

- Get Bob's ID, e.g. `bob@ve475.sjtu.edu.cn`
- Compute $g = e(H_1(\texttt{bob@ve475.sjtu.edu.cn}), Q)$
- Select a random $r$ in $\mathbb{Z}_q^*$
- Compute $t = m \oplus H_2(g^r)$
- Send the ciphertext $C = \langle [r]P, t \rangle$

## Decryption

Bob recovers Alice's message

Given a ciphertext $C = \langle [r]P, t \rangle$:

- Set $ID$ to `bob@ve475.sjtu.edu.cn`
- Compute $h = e(s_{ID}, [r]P)$
$$\begin{aligned} h &= e([s]H_1(ID), [r]P) = e(H_1(ID), P)^{sr} \\ &= e(H_1(ID), [s]P)^r = e(H_1(ID), Q)^r \\ &= g^r \end{aligned}$$
- Recover the message as
$$t \oplus H_2(h) = m \oplus H_2(g^r) \oplus H_2(g^r) = m$$

## Security

Basic remarks on the security:

- If $s$ can be recovered then all the secret keys can be revealed
- If $r$ can be computed from $[r]P$ then the message can be recovered
- The hash functions must be collision resistant
  e.g. $h = e(H_1(ID), [r]P)^s = g_p^s = g^r$. Neither $s$ nor $g^r$ is known but if we can find $s'$ such that $H_2(g_p^{s'}) = H_2(g^r)$ then $m$ can be recovered
- The TA must be trusted

## Attribute based cryptography

Identity Based Cryptography: public key generated from an ID

Attribute Based Cryptography: public key generated from attributes

Typical use:

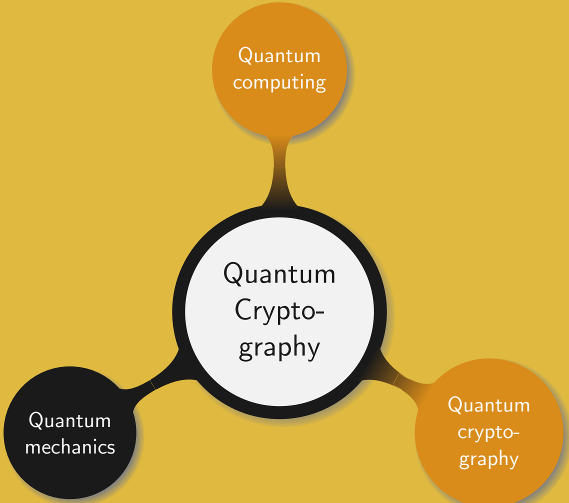
- Company: a class of employees is sent some encrypted information; no need to encrypt using the public key of each employee
- Social media: people can belong to many different groups and want to share information only with a certain group without encrypting a special version for each member of the group
- Broadcast encryption: different class of users have payed for different services

## Key points

- What is an elliptic curve?
- What is the main advantage of elliptic curves in cryptography?
- What is the most useful property of a pairing?
- Explain what Identity Based Cryptography and Attribute Based Cryptography are

## 9. Quantum Cryptography

## Chapter organisation



## Quantum mechanics

Basics on quantum mechanics:

- Physics at the atomic and subatomic levels
- Accurate and precise theory
- The state of the system is not given by a physical observation
- Impossible to know exactly the state of the system
- Probabilistic predictions can be made

## Formalism

Mathematical formulation:

- Every system is associated with a separable Hilbert space $H$
- A state of the system is represented by a unit vector in $H$
- The *Ket A* denoted $|A\rangle$ represents the column vector $A = a_1|e_1\rangle + a_2|e_2\rangle + \cdots + a_n|e_n\rangle$, where $|e_1\rangle, \ldots, |e_n\rangle$ form a basis for $H$

$$|A\rangle = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

- The *Bra B* denoted $\langle B|$ is the conjugate transpose of $|B\rangle$

$$\langle B| = \begin{pmatrix} b_1^* & b_2^* & \cdots & b_n^* \end{pmatrix}$$

## Formalism

Mathematical formulation:

- The *inner product* of $B$ and $A$ is

$$\langle B|A\rangle = b_1^* a_1 + b_2^* a_2 + \cdots + b_n^* a_n$$

- The *outer product* of $A$ and $B$ is the tensor product of $A$ and $B$ and is denoted

$$|A\rangle\langle B| = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \otimes \begin{pmatrix} b_1^* & b_2^* & \cdots & b_n^* \end{pmatrix} = \begin{pmatrix} a_1 b_1^* & a_1 b_2^* & \cdots & a_1 b_n^* \\ a_2 b_1^* & a_2 b_2^* & \cdots & a_2 b_n^* \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1^* & a_n b_2^* & \cdots & a_n b_n^* \end{pmatrix}$$

- An *observable* quantity is represented by an Hermitian matrix $M$

## Quantum Principles

Basic ideas behind quantum physics:

- $M$ can be unitarily diagonalized
- The possible outcomes of $M$ are its eigenvectors
- Its eigenvectors $|\phi_i\rangle$, $1 \leq i \leq n$, generate an orthogonal basis
- Any vector $|\psi\rangle$ can be written as a *superposition* of the $|\phi_i\rangle$
$$|\psi\rangle = c_1|\phi_1\rangle + \cdots + c_n|\phi_n\rangle$$
- A measurement of $M$ results in $|\phi_i\rangle$ with probability $|c_i|^2$
- Two quantum objects, whose states can only be described with reference to each other, are said to be *entangled*

## Example

For $|A\rangle = \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix}$, $\langle A|A\rangle = \begin{pmatrix} 1/2 & 1/2 & 1/2 & 1/2 \end{pmatrix} \cdot \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} = 1$, and

$$|A\rangle\langle A| = \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} \otimes \begin{pmatrix} 1/2 & 1/2 & 1/2 & 1/2 \end{pmatrix}$$

$$= \begin{pmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{pmatrix}.$$

## Example

Given two particles which can collapse in the states 0 or 1, the four possible outcomes are $|00\rangle,\ |01\rangle,\ |10\rangle,\ |11\rangle$.

The general state of the two particles is given by the superposition

$$|\psi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle, \text{ with } \sum_{i=0}^{3} |a_i|^2 = 1.$$

Some states might be written as a product of states for each particle. For instance

$$\frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right).$$

However in some other cases, such as $\frac{1}{\sqrt{2}}\left(|01\rangle + |10\rangle\right)$, it cannot be factorized. The particles are then said to be entangled.

## EPR paradox

Einstein Podolsky Rosen paper (1935):

- Two particles interact and get entangled
- They form a system which remains in this superposition until a measurement is performed
- The two particles travel far from each other

For instance if a measurement is realised on the first particle from the previous example (9.368) and the outcome is $|0\rangle$ then the second particle must be in state $|1\rangle$.

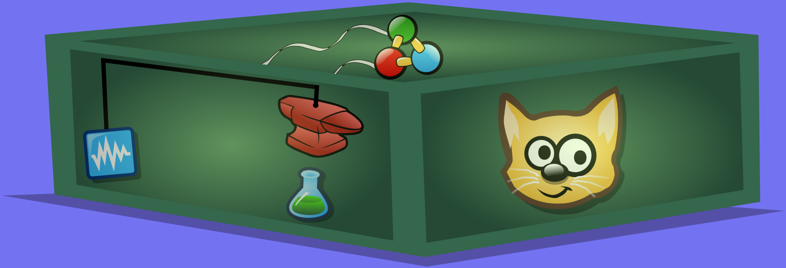
This idea conflicts with the theory of relativity which states that nothing can travel faster than the speed of light. In fact the measurement of $|0\rangle$ on the first particle implies probability 1 of getting $|1\rangle$ whatever the distance between the two particles.

## Schrödinger's cat

Basic idea: create a system where a radioactive atom is "entangled with a cat". If the atom decays and emit radiation some poison is released and the cat dies.

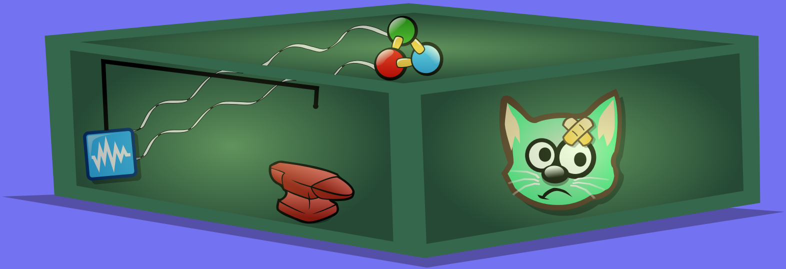
## Schrödinger's cat

Basic idea: create a system where a radioactive atom is "entangled with a cat". If the atom decays and emit radiation some poison is released and the cat dies.



## Schrödinger's cat

Basic idea: create a system where a radioactive atom is "entangled with a cat". If the atom decays and emit radiation some poison is released and the cat dies.



## Schrödinger's cat

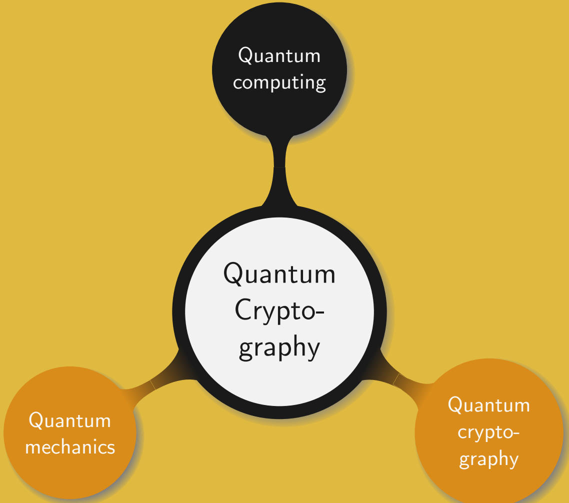
Basic idea: create a system where a radioactive atom is "entangled with a cat". If the atom decays and emit radiation some poison is released and the cat dies.

## Quantum teleportation

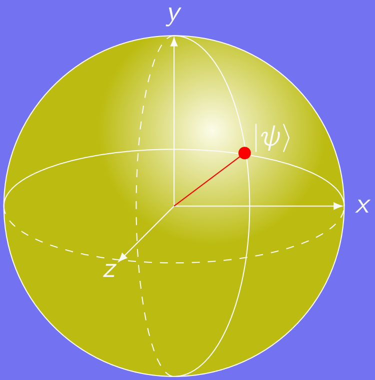
High level idea:

- Alice and Bob meet to construct an EPR pair $(A, B)$, Alice takes $A$ and Bob $B$
- Alice wants to share quantum information on a particle $C$ with Bob
- Alice creates an EPR pair $(A, C)$ such that $A$, $B$, and $C$ are now entangled
- Alice measures $A$ such that $B$ collapses in a state that “resembles” the state of $C$
- Using a classical channel Alice sends the state of $A$ to Bob
- When Bob knows the state of $A$ he can easily work out the state $C$ is in

## Chapter organisation



## Bit vs. Qubit

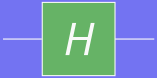


- A *qubit* is to a quantum computer, what a bit is to a classical computer
- A qubit can be in any superposition of the states $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- Several entangled qubits taken together form a *register*
- A *quantum gate* is a unitary transform on a fixed number of qubits
- A *quantum circuit* is a set of quantum gates linked together

## Quantum gates

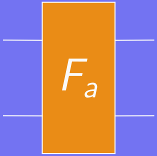
Hadamard transform:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$



Inverse the phase of the third state in a superposition of two qubits:

$$F_a = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

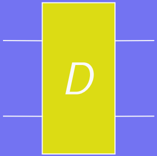


More generally $F_a|x\rangle = \begin{cases} -|a\rangle & \text{if } x = a \\ |x\rangle & \text{otherwise} \end{cases}$

## Quantum gates

Diffusion transform for a superposition of two qubits:

$$D = \frac{1}{2}\begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix}$$

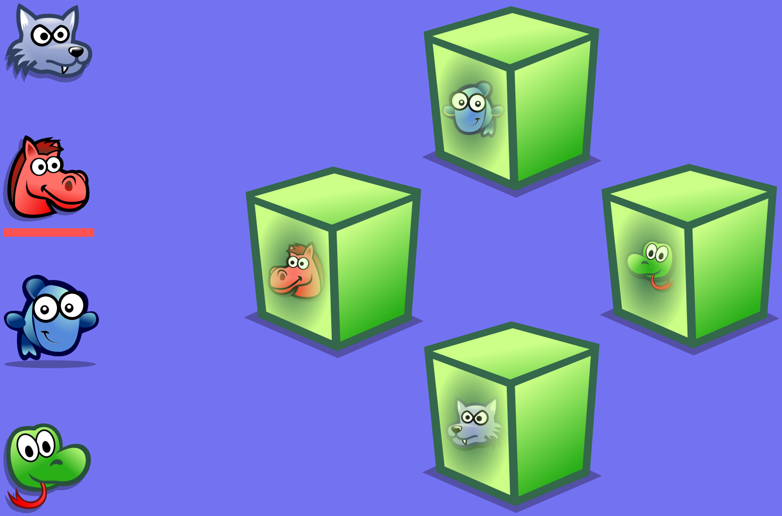


Calling the initial superposition of $n$ states $|\psi_0\rangle$ it is generalized as

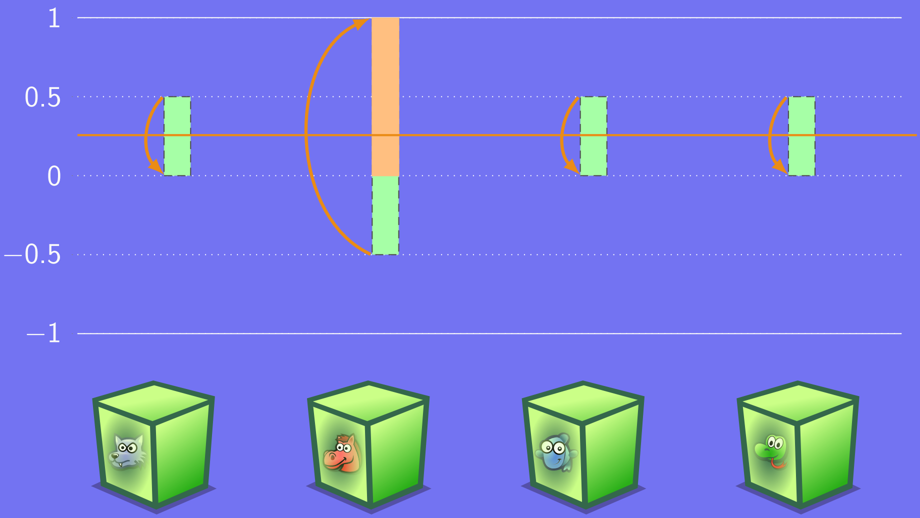
$$D = 2|\psi_0\rangle\langle\psi_0| - I_n,$$

where $I_n$ is the identity matrix of dimension $n$.

## A quantum search



## Grover's algorithm



## Finding the horse

For the sake of simplicity we define *wolf* $= |00\rangle$, *horse* $= |01\rangle$, *fish* $= |10\rangle$, and *snake* $= |11\rangle$. Therefore only two qubits are needed.

We start with the superposition

$$\frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) = \frac{1}{2}\begin{pmatrix}1\\1\\1\\1\end{pmatrix}.$$

This is achieved by setting the two qubits to $|0\rangle$ an applying an Hadamard transform to each of them:

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}.$$

## Finding the horse

Then applying the transform $F_{horse}$ yields

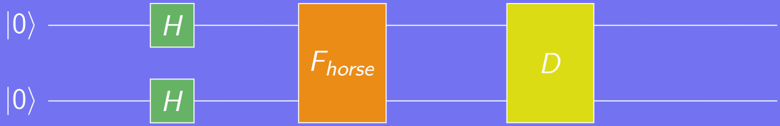
$$\frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix}.$$

Finally after applying $D$ we get

$$\frac{1}{4}\begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

## Quantum circuit

In the case of four elements this can be represented using the following quantum circuit.



This can easily be extended to a set of $n$ elements, by using a register of $\log_2 n$ qubits. Hadamard transforms are applied to get the superposition

$$|s\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} |i\rangle.$$

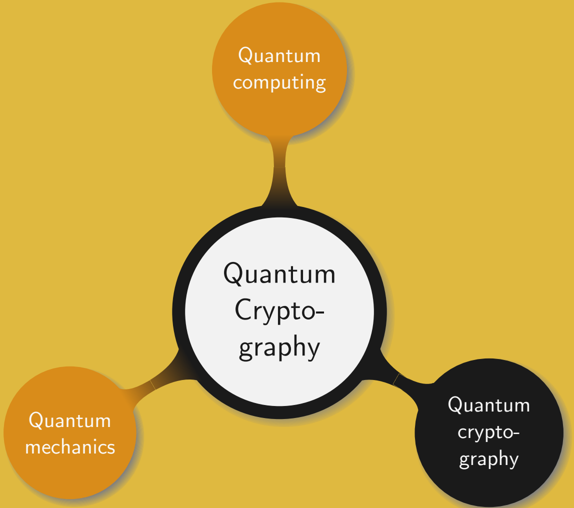
The horse can be found by repetitively applying the $F_{horse}$ and $D$ transforms $O(\sqrt{n})$ times.

## Quantum computing and cryptography

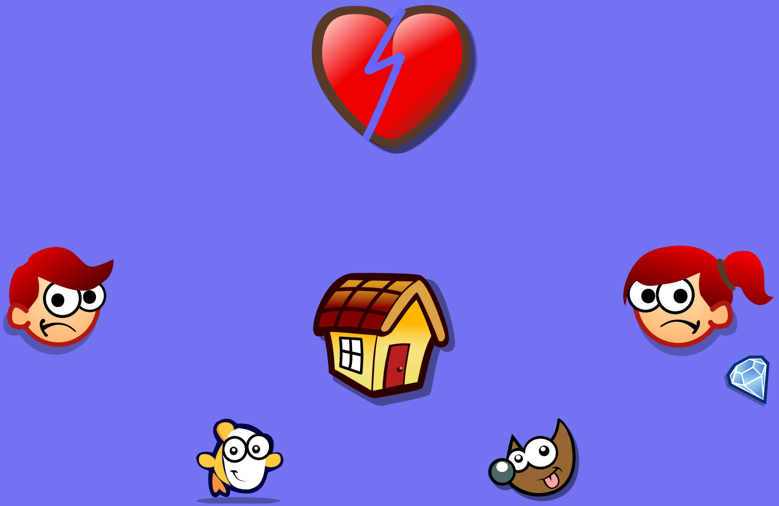
A few additional remarks:

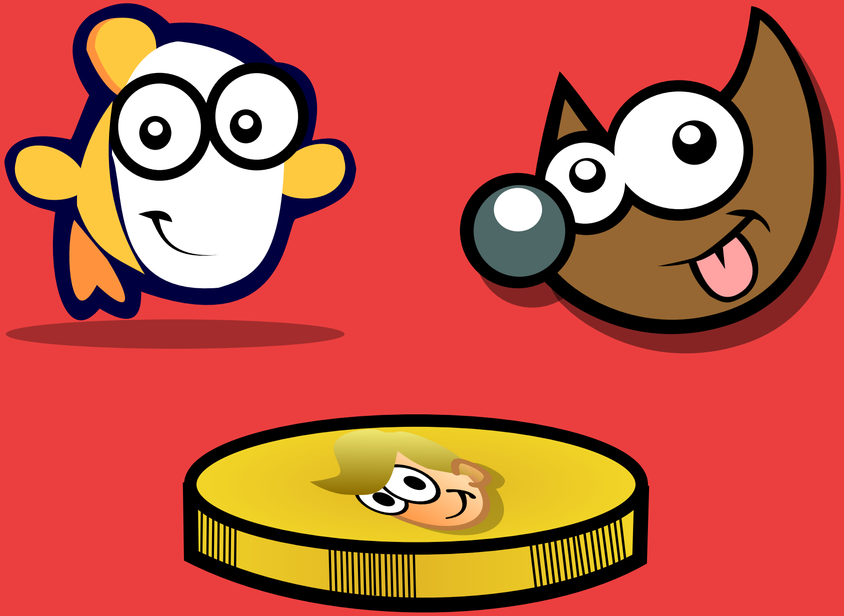
- Grover's algorithm can be used to find collisions in $\mathcal{O}(n^{1/3})$ queries
- Quantum algorithms solving the RSA and the DLP in polynomial time exist
- The actual record, established in 2012, is $56153 = 233 \cdot 241$
- Several hard problems in multivariate and lattice cryptography cannot be solved in polynomial time on a quantum computer

## Chapter organisation



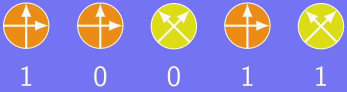
## The true story of Alice and Bob



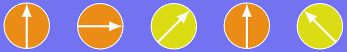


## Quantum bit commitment

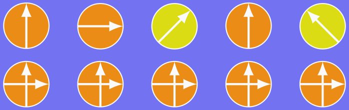
1. Pick $n$ elements in $\{0, 1\}$ and $n$ basi
at random

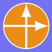


2. Send to Bob



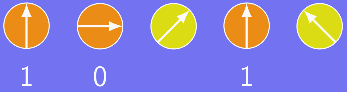
3. Commit to $b$ by choosing a basis for the measurements



4. Send to Alice with $b =$ 

1 0 1 1 1

5. Only verify qubits in base $b$



## Security

Security of the protocol:

- Alice generates the 0,1 and the basis but has no idea on $b$, so she cannot cheat
- If Bob has access to a large quantum memory he can copy the qubits, measure them in a basis and their copy in the other basis
- Qubits are very hard to store, so it is a fair assumption to assume that Bob cannot store the $n$ qubits
- As a measurement destroys the information he cannot measure again in another basis and as such he cannot cheat

## Bit commitment

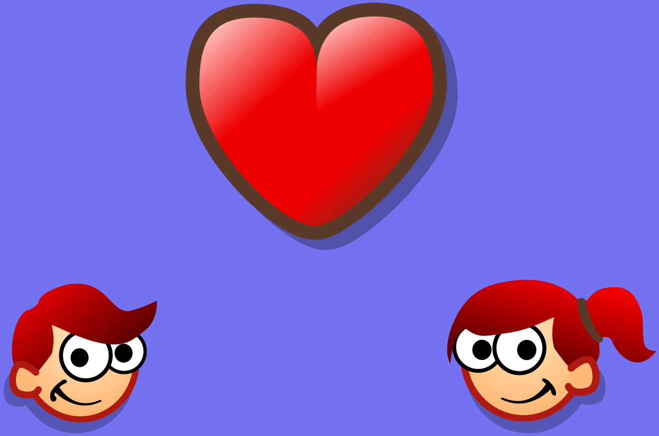
Bit commitment protocols are very simple, and can easily be realised without the help of quantum cryptography.

Example. Simple bit commitment protocol:

- Bob generates a 100-bit long string
- He appends his bit $b$ and another 100-bit long string
- He sends the hash of the 201-bit long string to Alice
- To reveal $b$ Bob sends the 201-bit long string

## The end...

While figuring out a bit commitment protocol Alice and Bob forgot why they wanted to get a divorce and they fell in love again...

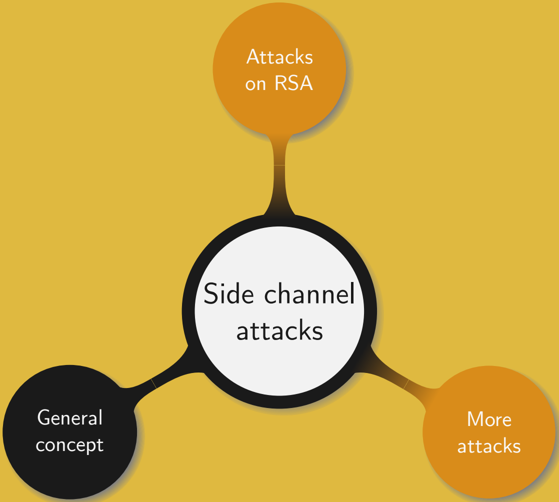


## Key points

- For two particles what does it mean to be entangled?
- What is a qubit?
- What is the advantage of quantum computing over classical computing?
- What is a bit commitment protocol?

## 10. Side channel attacks

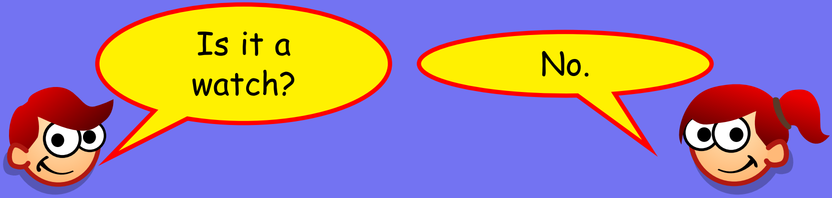
## Chapter organisation

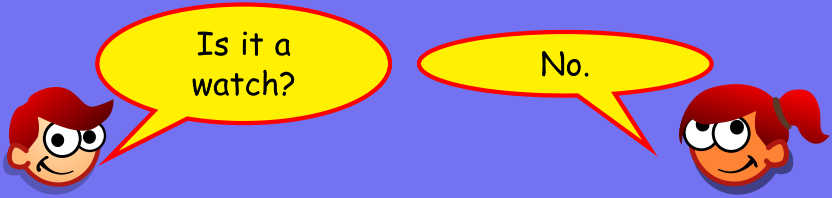


## Basic idea

Curious Bob want to know what is his present

## Basic idea




Is it a watch?
No.

## Side channel attacks

Computers leak information when performing an operation:

- Amount of power used
- Time spent
- Area of the memory used
- Electromagnetic radiations
- Sounds (hard disk, beep...)
- Frequency of sending packets on the network

## General remarks

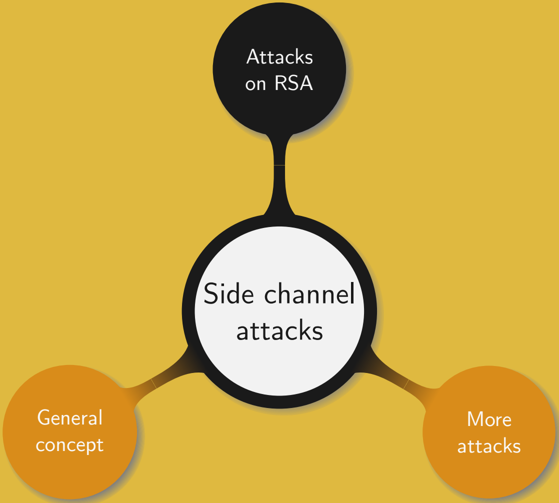
A few important notes:

- Side channel attacks apply to any protocol
- A secure cipher with no attack on the protocol is not immune to side channel attacks
- Not many way to get protected

Example. SSH is a secure way to connect to a remote computer. If a user authenticate using a password, the time between each keystroke, or packet sent on the network, can be analysed and the password recovered.

An audio recording of someone typing on a keyboard is enough to know what he is writing.

## Chapter organisation



## RSA implementations

To break RSA the goal is to find the secret key:

- When a user decrypts a message
- When a user signs a message

The attacker can access to the host device to:

- Run some malicious code
- Perform measurements

As running RSA leaks information the attacker only needs to read it and then perform some analysis in order to interpret it.

## Timing attack

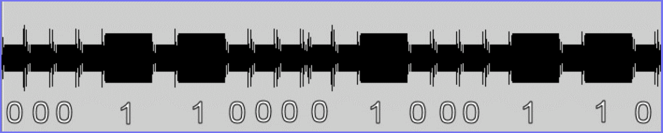
General approach:

- RSA decryption/signature uses the square and multiply algorithm (3.172)
- On a 0 only a squaring occurs
- On a 1 a squaring and a multiplication occur

The mean not being precise enough the variance is used to analyse a large number of decryption requests. The attacker then uses the fact that the variance of the sum of two independent random processes is the sum of their respective variance. He gains little information on the secret key but he reuses it to perform more accurate measurements and in the end he is able to totally recover the key.

## Power analysis attack

In this attack the key idea is to observe the power consumption of the computer when decrypting or signing a message.



On a 1 both a square and a multiply are carried out. When only a squaring occurs the power consumption is much lower. This attack is clearly much more powerful and efficient than the previous one. In fact no more than one decryption is necessary to recover the secret key.

We know present an algorithm that performs modular exponentiation without leaking much information.

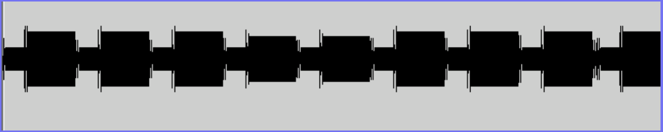
## Montgomery's ladder technique

Algorithm. (*Modular exponentiation*)

```
Input   : $m$ an integer, $d = (d_{k-1} \ldots d_0)_2$ and $n$ two positive integers
Output: $x = m^d \bmod n$
power1 ← m; power2 ← m^2;
for $i \leftarrow k-2$ to 0 do
    if $d_i = 0$ then
        $power2 \leftarrow (power1 \cdot power2) \bmod n$;
        $power1 \leftarrow power1^2 \bmod n$;
    else
        $power1 \leftarrow (power1 \cdot power2) \bmod n$;
        $power2 \leftarrow power2^2 \bmod n$;
    end if
end for
return $power1$
```

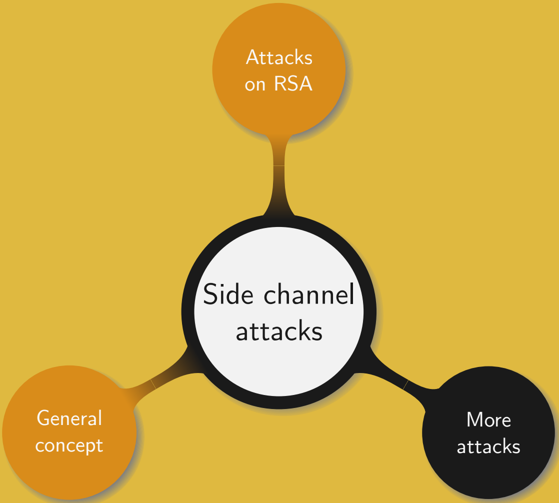
## Preventing attacks on RSA

The previous algorithm has the advantage of performing both multiplication and squaring whatever the bit considered. Therefore monitoring the power consumption would not bring any information on the secret key as it would look as on the following picture.



Note that the Montgomery's ladder technique is still vulnerable to cache timing attacks. Indeed an attacker could measure the time necessary to access the memory, and as this depends on which variable is used he could recover some information on the bits composing the secret key.

## Chapter organisation



## Reverse engineering

Extract information on the design and process of a program:

- Get the binary file
- Disassemble it
- Understand the generated assembly code
- Extract some important information

*Never store a secret key in a binary, “everybody” can retrieve it*

## Background on memory

Page sharing:

- Share parts of the memory between processes
- Avoid replicated copies of identical content
- Pages are read-only
- On a write request, copy the page onto a new writable location

Cache structure in modern processors:

- Each core has two levels of cache L1 and L2
- A third level L3 is shared among all the cores
- Removing data from L3 also flushes it from both L1 and L2
- The closer from the CPU the faster to retrieve data

A cache timing attack measures how long it takes the CPU to fetch the data. This leaks information on what operation is performed.

## L3 cache side channel attack

High level idea applied by the attacker:

- Use `mmap` to map the victim's executable file into the attacker's address space
- Mark up memory lines related to specific operations
- Flush the memory line from the L3 cache and wait
- Call the memory line and measure how long it takes it load it
- Slow loading: the line was not called by the victim
- Fast loading: the line is in the cache, meaning it was used

It is impossible to prevent this attack on a multi-user system as it is inherent to the implementation of the X86 architecture. Only a hardware fix could solve this weakness.

## Preventing side channel attacks

Simple conclusion:

- It is impossible to prevent all the kinds of side channel attack
- A system can never be 100% secure

What can be done:

- Chose secure protocols
- Select secure libraries
- GMP implements function intended for a cryptographic use
- Such functions are slower but more resistant to side channel attacks

## A final example

VMWare View is a popular remote desktop protocol. VMWare recommends to switch from AES-128 to SALSA20-256 for the “best user experience”.

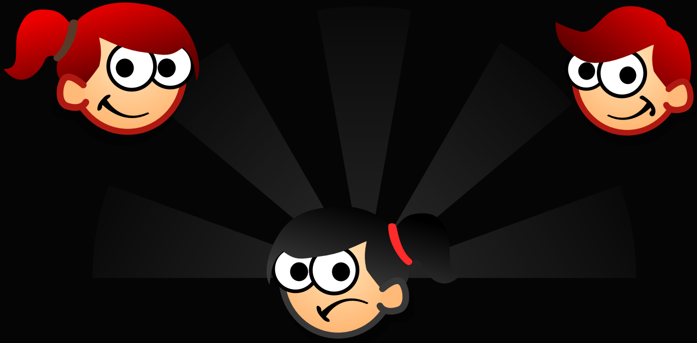
A closer look at the specs shows that AES-128 refers to AES-128-GCM, which includes AES and message authentication. On the other hand SALSA20-256 refers to SALSA20-256-Round12, which does not feature any message authentication.

Although SALSA20 has speed and security advantages over AES an attacker can easily forge packets if it is not used in conjunction with with message authentication.

*Is it worth sacrificing security for the sake of speed?*

## Key points

- Explain what are side channel attacks
- List two examples of side channel attacks
- How to prevent side channel attacks?
- Can a system be made fully secure?



Thank you, enjoy the Summer break!

## References I

1.31 https://www.xkcd.com/538/

1.47 Simon Singh, *The Code Book – How to make it, break it, hack it, crack it?*

1.61 https://cdn.globalauctionplatform.com/7187abcf-14de-4d26-9a48-a48e012a3bd3/1f194fa9-87f4-45cd-b6db-a48e012de5c7/original.jpg

2.86 https://upload.wikimedia.org/wikipedia/commons/5/56/Tux.jpg

2.86 https://upload.wikimedia.org/wikipedia/commons/f/f0/Tux_ecb.jpg

2.86 https://upload.wikimedia.org/wikipedia/commons/a/a0/Tux_secure.jpg

2.89 https://www.xkcd.com/221/

10.401 G. Hollestelle, W. Burgers, J. den Hartog, *Power analysis on smartcard algorithms using simulation*

10.403 G. Hollestelle, W. Burgers, J. den Hartog, *Power analysis on smartcard algorithms using simulation*