DD2448 Foundations of Cryptography
Lecture 1

Douglas Wikström
KTH Royal Institute of Technology
dog@kth.se

January 18, 2016

Introduction and Administration

Information About the Course

- Oral information given and agreements made during lectures.
- Read at: `https://www.kth.se/social/course/DD2448`
- Read your KTH email: `<username>@kth.se`

If this fails, then email `dog@kth.se`.

Use `DD2448` in the subject line.

What is cryptography?

Cryptography is concerned with the conceptualization, definition, and construction of computing systems that address security concerns.

– Oded Goldreich, Foundations of Cryptography, 1997

Applications of Cryptography

Historically.

- Military and diplomatic secret communication.
- Communication between banks, e.g., credit card transactions.

Modern Time.

- Protecting satellite TV from leaching.
- Secrecy and authenticity on the Internet, mobile phones, etc.
- Credit cards.

Applications of Cryptography

Today.

- Distributed file systems, authenticity of blocks in bit torrents, anonymous remailers, Tor-network, etc.
- RFID tags, Internet banking, Försäkringskassan, Skatteverket, "e-legitimation".

Future.

- Secure distributed computing (multiparty computation): election schemes, auctions, secure cloud computing, etc.
- Variations of signatures, cryptosystem, and other primitives with special properties, e.g., group signatures, identity based encryption, etc.

Goal

The goal of the course is to

- give an overview of modern cryptography

in order that students should

- know how to evaluate and, to some extent, create cryptographic constructions, and
- to be able to read and to extract useful information from research papers in cryptography.

Prerequisites

- *DD1352 Algorithms, data structures and complexity*, or *DD2354 Algorithms and complexity*.
- Knowledge of mathematics and theory of algorithms corresponding to the required courses of the D or F-programmes at KTH.

Tentative Plan of Content (1/2)

- Administration, introduction, classical cryptography.
- Symmetric ciphers, substitution-permutation networks, linear cryptanalysis, differential cryptanalysis.
- AES, Feistel networks, DES, modes of operations, DES-variants.
- Entropy and perfect secrecy.
- Repetition of elementary number theory,
- Public-key cryptography, RSA, primality testing, textbook RSA, semantic security.

Tentative Plan of Content (2/2)

- RSA in ROM, Rabin, discrete logarithms, Diffie-Hellman, El Gamal.
- Security notions of hash functions, random oracles, iterated constructions, SHA, universal hash functions.
- Message authentication codes, identification schemes, signature schemes, PKI.
- Elliptic curve cryptography.
- Pseudorandom generators.
- Guest lecture.
- Make-up time and/or special topic.

Working Example

Throughout the course we will use electronic voting systems to motivate the notions introduced and how to use them.

Course Requirements

Group project about authentication.
Students are divided into groups of three. If the number of students is not divisible by three, then one or two groups will have four members.

- Describe and provide a security analysis of a way to authenticate a voter in an Internet voting system (in abstract form).

  Judged by quality of description and level of rigor in the analysis. (20P)

- Study a real world example of the abstract description, i.e., study how the abstract description is turned into a specification.

  Judged by level of detail and relevance of topics. (20P)

Course Requirements

Group project about authentication (cont.).

- Implement an HTML/CSS/JavaScript client that uses the real world authentication scheme.

  Judged by how complete and robust it is. (20P)

- Identify security flaws at any level in the real world authentication scheme, theoretical, in publicly available code fragments, or in solutions of other groups.

  Judged by the quality and number of observations. (20P)

Course Requirements

Group project about authentication (cont.).

- Code is provided for encryption in client (will be discussed in class).
- Pluggable server will be provided for server side. Students write plugin.

Course Requirements

Homework 1-2. Each homework is a set of problems giving I-points and T-points ($I \geq 10$ and $I + T \geq 100$).

- Solved in groups of up to three students, which may differ for each homework.
- Only informal discussions are allowed.
- Each student writes and submits his own solution.

Detailed rules and advice are found on the course homepage.

Only complete homeworks can be replaced following years.

(Less than previous years, but more than two such homeworks.)

Qualify for Oral Exam (Possibly)

There may be a multiple choice exam during one of the lectures with a pass/fail grade to qualify for the oral exam.

Course Requirements

Oral Exam. Purpose is to give a fair grade.

Discussion starts from submitted solutions and the project to ensure that the grading corresponds to the skills of the student, but can move on to any subject covered in the course.

- For each problem or project I-points or T-points may be added or removed from the original grading depending on the understanding shown by the student.
- The updated number of points of a problem is never negative and never more than the nominal maximum number of points of the problem stated in the homework.
- A single O-point is awarded after passing the exam.

Deadlines

The deadlines in this course are given on the homepage and are strict. Late solutions are awarded zero points.

However, if practically possible, then we negotiate the deadlines to not conflict unnecessarily with other courses.

Grading

To earn a given grade the requirements of all lower grades must be satisfied as well, with $A = I + T + P + O$.

Grade	Requirements
E	$I \geq 30$, $T \geq 40$, $P \geq 30$, and $O \geq 1$.
D	$A \geq 120$.
C	$A \geq 140$ and $P \geq 50$.
B	$A \geq 170$.
A	$A \geq 210$ and $P \geq 60$.

Kattis

Kattis is a judging server for programming competitions and for grading programming assignments. We use it for all isolated exercises where code is submitted as a solution.

We assume that your Kattis id is the same as your KTH user name. If this is not the case, then email us your Kattis user name and use the subject `Krypto16 Kattis`.

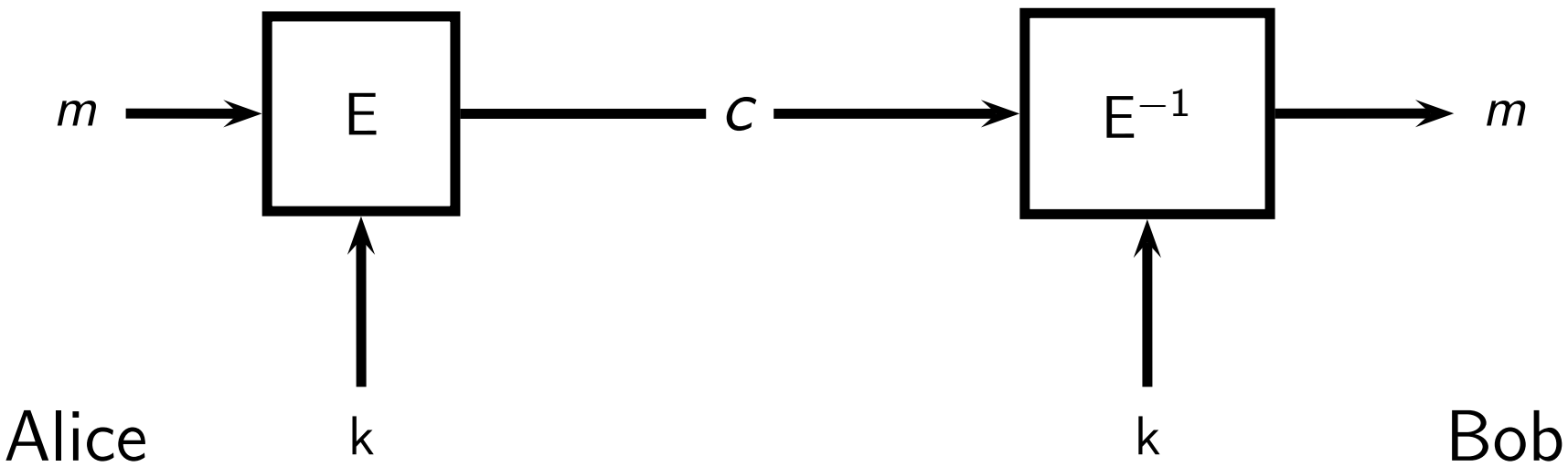
Latex

- Latex is the standard typesetting tool for mathematics.
- It is the fastest way to produce mathematical writing. **You must use it to typeset your solutions.**
- The best way to learn it is to read: `http://tobi.oetiker.ch/lshort/lshort.pdf`

Introduction to Ciphers

Cipher (Symmetric Cryptosystem)

$c = \mathsf{E}_k(m)$ $\qquad$ $m = \mathsf{E}_k^{-1}(c)$



Cipher (Symmetric Cryptosystem)

Definition. A cipher (symmetric cryptosystem) is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathsf{E}, \mathsf{E}^{-1})$, where

Cipher (Symmetric Cryptosystem)

Definition. A cipher (symmetric cryptosystem) is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathsf{E}, \mathsf{E}^{-1})$, where

- Gen is a probabilistic **key generation algorithm** outputting keys from a key space $\mathcal{K}$,

Cipher (Symmetric Cryptosystem)

Definition. A cipher (symmetric cryptosystem) is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathsf{E}, \mathsf{E}^{-1})$, where

- Gen is a probabilistic **key generation algorithm** outputting keys from a key space $\mathcal{K}$,
- $\mathcal{P}$ is a **set of plaintexts**,

Cipher (Symmetric Cryptosystem)

Definition. A cipher (symmetric cryptosystem) is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathsf{E}, \mathsf{E}^{-1})$, where

- Gen is a probabilistic **key generation algorithm** outputting keys from a key space $\mathcal{K}$,
- $\mathcal{P}$ is a **set of plaintexts**,
- E is a deterministic **encryption algorithm**, and

Cipher (Symmetric Cryptosystem)

Definition. A cipher (symmetric cryptosystem) is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathsf{E}, \mathsf{E}^{-1})$, where

- Gen is a probabilistic **key generation algorithm** outputting keys from a key space $\mathcal{K}$,
- $\mathcal{P}$ is a **set of plaintexts**,
- E is a deterministic **encryption algorithm**, and
- E^{-1} is a deterministic **decryption algorithm**,

Cipher (Symmetric Cryptosystem)

Definition. A cipher (symmetric cryptosystem) is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathsf{E}, \mathsf{E}^{-1})$, where

- Gen is a probabilistic **key generation algorithm** outputting keys from a key space $\mathcal{K}$,
- $\mathcal{P}$ is a **set of plaintexts**,
- E is a deterministic **encryption algorithm**, and
- E^{-1} is a deterministic **decryption algorithm**,

such that $\mathsf{E}_\mathsf{k}^{-1}(\mathsf{E}_\mathsf{k}(m)) = m$ for every message $m \in \mathcal{P}$ and $\mathsf{k} \in \mathcal{K}$. The set $\mathcal{C} = \{\mathsf{E}_\mathsf{k}(m) \mid m \in \mathcal{P} \wedge \mathsf{k} \in \mathcal{K}\}$ called the **set of ciphertexts**.

Attacks

Throughout the course we consider various attacks on cryptosystems. With small changes, these attacks make sense both for symmetric and asymmetric cryptosystems.

- Ciphertext-only attack.
- Known-plaintext attack
- Chosen-plaintext attack
- Chosen-ciphertext attack

Ceasar Cipher (Shift Cipher)

Consider English, with alphabet A-Z_, where _ denotes space, thought of as integers 0-26, i.e., $\mathbb{Z}_{27}$

- **Key.** Random letter $k \in \mathbb{Z}_{27}$.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where $c_i = m_i + k \bmod 27$.
- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where $m_i = c_i - k \bmod 27$.

Ceasar Cipher Example

Encoding.

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	_
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Key: $G = 6$

Plaintext.	B	R	I	B	E	_	L	U	L	A	_	T	O	_	B	U	Y	_	J	A	S
Plaintext.	01	17	08	01	04	26	11	20	11	00	26	19	14	26	01	20	24	26	09	00	18
Ciphertext.	07	23	14	07	10	05	17	26	17	06	05	25	20	05	07	26	03	05	15	06	24
Ciphertext.	H	X	O	H	K	F	R	_	R	G	F	Z	U	F	H	_	D	F	P	G	Y

Statistical Attack Against Ceasar (1/3)

Decrypt with all possible keys and see if some English shows up, or more precisely...

Statistical Attack Against Ceasar (2/3)

Written English Letter Frequency Table $F[\cdot]$.

A	0.072	J	0.001	S	0.056
B	0.013	K	0.007	T	0.080
C	0.024	L	0.035	U	0.024
D	0.037	M	0.021	V	0.009
E	**0.112**	N	0.059	W	0.021
F	0.020	O	0.066	X	0.001
G	0.018	P	0.017	Y	0.017
H	0.054	Q	0.001	Z	0.001
I	0.061	R	0.053	_	**0.120**

Note that the same frequencies appear in a ciphertext of written English, but in shifted order!

Statistical Attack Against Ceasar (3/3)

- Check that the plaintext of our ciphertext has similar frequencies as written English.
- Find the key k that maximizes the inner product $T(\mathrm{E}_\mathrm{k}^{-1}(C)) \cdot F$, where $T(s)$ and F denotes the frequency tables of the string s and English.

This usually gives the correct key k.

Affine Cipher

Affine Cipher.

- **Key.** Random pair $\mathrm{k} = (a, b)$, where $a \in \mathbb{Z}_{27}$ is relatively prime to 27, and $b \in \mathbb{Z}_{27}$.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where $c_i = am_i + b \bmod 27$.
- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where $m_i = (c_i - b)a^{-1} \bmod 27$.

Substitution Cipher

Ceasar cipher and affine cipher are examples of substitution ciphers.

Substitution Cipher.

- **Key.** Random permutation $\sigma \in S$ of the symbols in the alphabet, for some subset S of all permutations.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where $c_i = \sigma(m_i)$.
- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where $m_i = \sigma^{-1}(c_i)$.

Digrams and Trigrams

- A digram is an ordered pair of symbols.
- A trigram is an ordered triple of symbols.
- It is useful to compute frequency tables for the most frequent digrams and trigrams, and not only the frequencies for individual symbols.

Generic Attack Against Substitution Cipher

1. Compute symbol/digram/trigram frequency tables for the candidate language and the ciphertext.
2. Try to match symbols/digrams/trigrams with similar frequencies.
3. Try to recognize words to confirm your guesses (we would use a dictionary (or Google!) here).
4. Backtrack/repeat until the plaintext can be guessed.

This is hard when several symbols have similar frequencies. A large amount of ciphertext is needed. How can we ensure this?

Vigénère

Vigénère Cipher.

- **Key.** $\mathsf{k} = (\mathsf{k}_0, \ldots, \mathsf{k}_{l-1})$, where $\mathsf{k}_i \in \mathbb{Z}_{27}$ is random.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where $c_i = m_i + \mathsf{k}_{i \bmod l} \bmod 27$.
- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where $m_i = c_i - \mathsf{k}_{i \bmod l} \bmod 27$.

Vigénère

Vigénère Cipher.

- **Key.** $\mathsf{k} = (\mathsf{k}_0, \ldots, \mathsf{k}_{l-1})$, where $\mathsf{k}_i \in \mathbb{Z}_{27}$ is random.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where $c_i = m_i + \mathsf{k}_{i \bmod l} \bmod 27$.
- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where $m_i = c_i - \mathsf{k}_{i \bmod l} \bmod 27$.

We could even make a variant of Vigénère based on the affine cipher, **but is Vigénère really any better than Ceasar?**

Attack Vigénère (1/2)

Index of Coincidence.

- Each probability distribution $p_1, \ldots, p_n$ on n symbols may be viewed as a point $p = (p_1, \ldots, p_n)$ on a $n-1$ dimensional hyperplane in $\mathbb{R}^n$ orthogonal to the vector $\overline{1}$
- Such a point $p = (p_1, \ldots, p_n)$ is at distance $\sqrt{F(p)}$ from the origin, where $F(p) = \sum_{i=1}^{n} p_i^2$.
- It is clear that p is closest to the origin, when p is the uniform distribution, i.e., when $F(p)$ is minimized.
- $F(p)$ is invariant under permutation of the underlying symbols $\longrightarrow$ tool to check if a set of symbols is the result of *some* substitution cipher.

Attack Vigénère (2/2)

1. For $l = 1, 2, 3, \ldots$, we form

$$\begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{l-1} \end{pmatrix} = \begin{pmatrix} c_0 & c_l & c_{2l} & \cdots \\ c_1 & c_{l+1} & c_{2l+1} & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ c_{l-1} & c_{2l-1} & c_{3l-1} & \cdots \end{pmatrix}$$

   and compute $f_l = \frac{1}{l} \sum_{i=0}^{l-1} F(C_i)$.

2. The local maximum with smallest l is probably the right length.
3. Then attack each C_i separately to recover k_i, using the attack against the Ceasar cipher.

Hill Cipher

Hill Cipher.

- **Key.** k $= A$, where A is an invertible $l \times l$-matrix over $\mathbb{Z}_{27}$.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where (computed modulo 27):

$$(c_{i+0}, \ldots, c_{i+l-1}) = (m_{i+0}, \ldots, m_{i+l-1})A \ .$$

- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where (computed modulo 27):

$$(m_{i+0}, \ldots, m_{i+l-1}) = (c_{i+0}, \ldots, c_{i+l-1})A^{-1} \ .$$

for $i = 1, l+1, 2l+1, \ldots$

Hill Cipher

Hill Cipher.

- **Key.** k $= A$, where A is an invertible $l \times l$-matrix over $\mathbb{Z}_{27}$.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where (computed modulo 27):

$$(c_{i+0}, \ldots, c_{i+l-1}) = (m_{i+0}, \ldots, m_{i+l-1})A \ .$$

- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where (computed modulo 27):

$$(m_{i+0}, \ldots, m_{i+l-1}) = (c_{i+0}, \ldots, c_{i+l-1})A^{-1} \ .$$

for $i = 1, l+1, 2l+1, \ldots$

The Hill cipher is easy to break using a known plaintext attack.

Permutation Cipher (Transposition Cipher)

The permutation cipher is a special case of the Hill cipher.

Permutation Cipher.

- **Key.** Random permutation $\pi \in S$ for some subset S of the set of permutations of $\{0, 1, 2, \ldots, l-1\}$.
- **Encrypt.** Plaintext $m = (m_1, \ldots, m_n) \in \mathbb{Z}_{27}^n$ gives ciphertext $c = (c_1, \ldots, c_n)$, where $c_i = m_{\lfloor i/l \rfloor + \pi(i \bmod l)}$.
- **Decrypt.** Ciphertext $c = (c_1, \ldots, c_n) \in \mathbb{Z}_{27}^n$ gives plaintext $m = (m_1, \ldots, m_n)$, where $m_i = c_{\lfloor i/l \rfloor + \pi^{-1}(i \bmod l)}$.

Last Lecture: Simple Ciphers

- Caesar cipher and affine cipher: $m_i \mapsto am_i + b$.
- Substitution cipher: $m_i \mapsto \sigma(m_i)$.
- Vigénère cipher: $m_i \mapsto m_i + k_{i \bmod l}$.
- Hill cipher (linear map):

$$(m_1, \ldots, m_l) \mapsto A(m_1, \ldots, m_l)$$

- Transposition cipher (permutation):

$$(m_1, \ldots, m_l) \mapsto (m_{\pi(1)}, \ldots, m_{\pi(l)})$$

Substitution-Permutation Networks

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

- A good cipher is one where each key gives a **randomly chosen permutation** of $\{0,1\}^n$.

  Why is this not possible?

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

- A good cipher is one where each key gives a **randomly chosen permutation** of $\{0,1\}^n$.

  Why is this not possible?

- The representation of a single typical function $\{0,1\}^n \to \{0,1\}^n$ requires roughly $n2^n$ bits ($147 \times 10^{6 \cdot 3}$ for $n = 64$)

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

- A good cipher is one where each key gives a **randomly chosen permutation** of $\{0,1\}^n$.

  Why is this not possible?

- The representation of a single typical function $\{0,1\}^n \to \{0,1\}^n$ requires roughly $n2^n$ bits ($147 \times 10^{6 \cdot 3}$ for $n = 64$)

- What should we look for instead?

Something Smaller

Idea. Compose smaller permutations into a large one. Mix the components "thoroughly".

Something Smaller

Idea. Compose smaller permutations into a large one. Mix the components "thoroughly".

Shannon (1948) calls this:

- **Diffusion.** "In the method of diffusion the statistical structure of M which leads to its redundancy is dissipated into long range statistics..."
- **Confusion.** "The method of confusion is to make the relation between the simple statistics of E and the simple description of K a very complex and involved one."

Substitution-Permutation Networks (1/2)

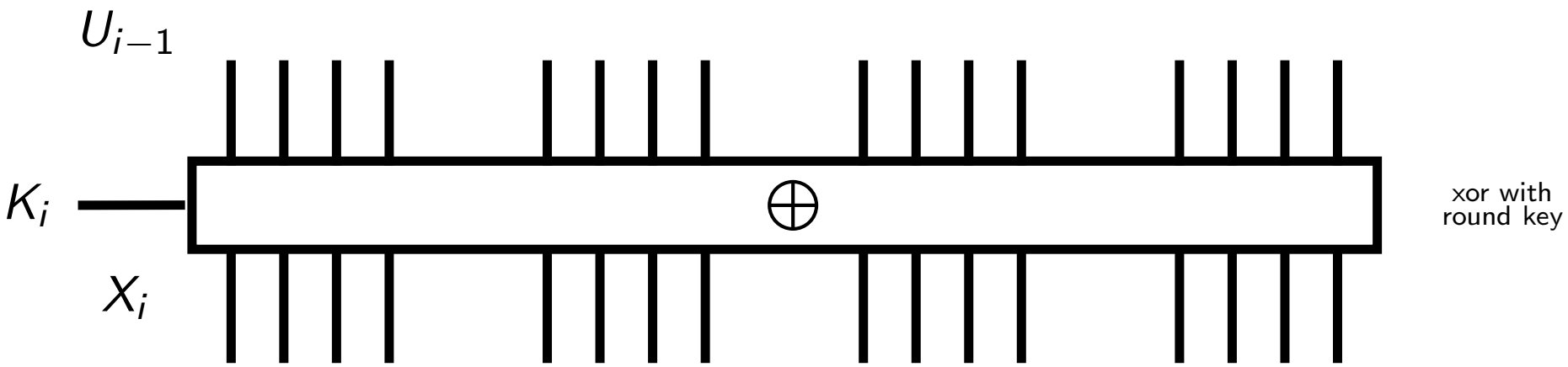
- **Block-size.** We use a block-size of $n = \ell \times m$ bits.
- **Key Schedule.** Round r uses its own round key K_r derived from the key K using a key schedule.
- **Each Round.** In each round we invoke:
  1. **Round Key.** xor with the round key.
  2. **Substitution.** ℓ substitution boxes each acting on one m-bit word (m-bit S-Boxes).
  3. **Permutation.** A permutation π_i acting on $\{1, \ldots, n\}$ to reorder the n bits.

Substitution-Permutation Networks (2/2)

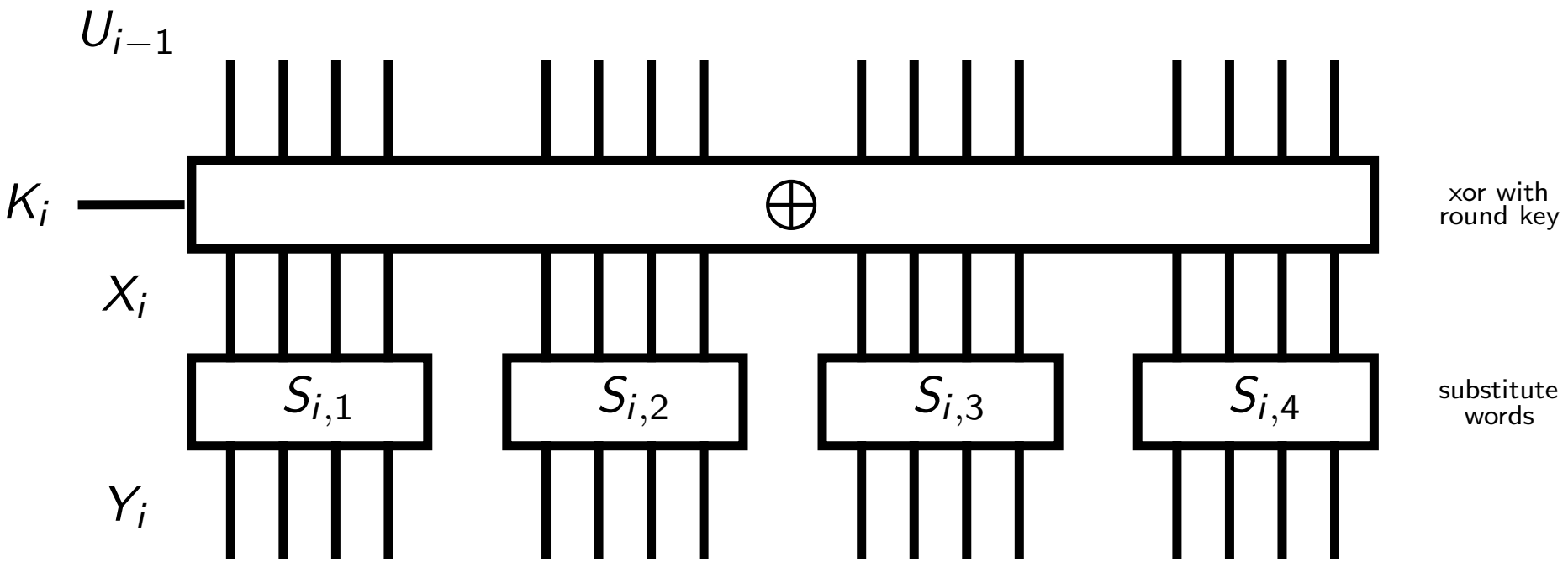
U_{i-1}

K_i

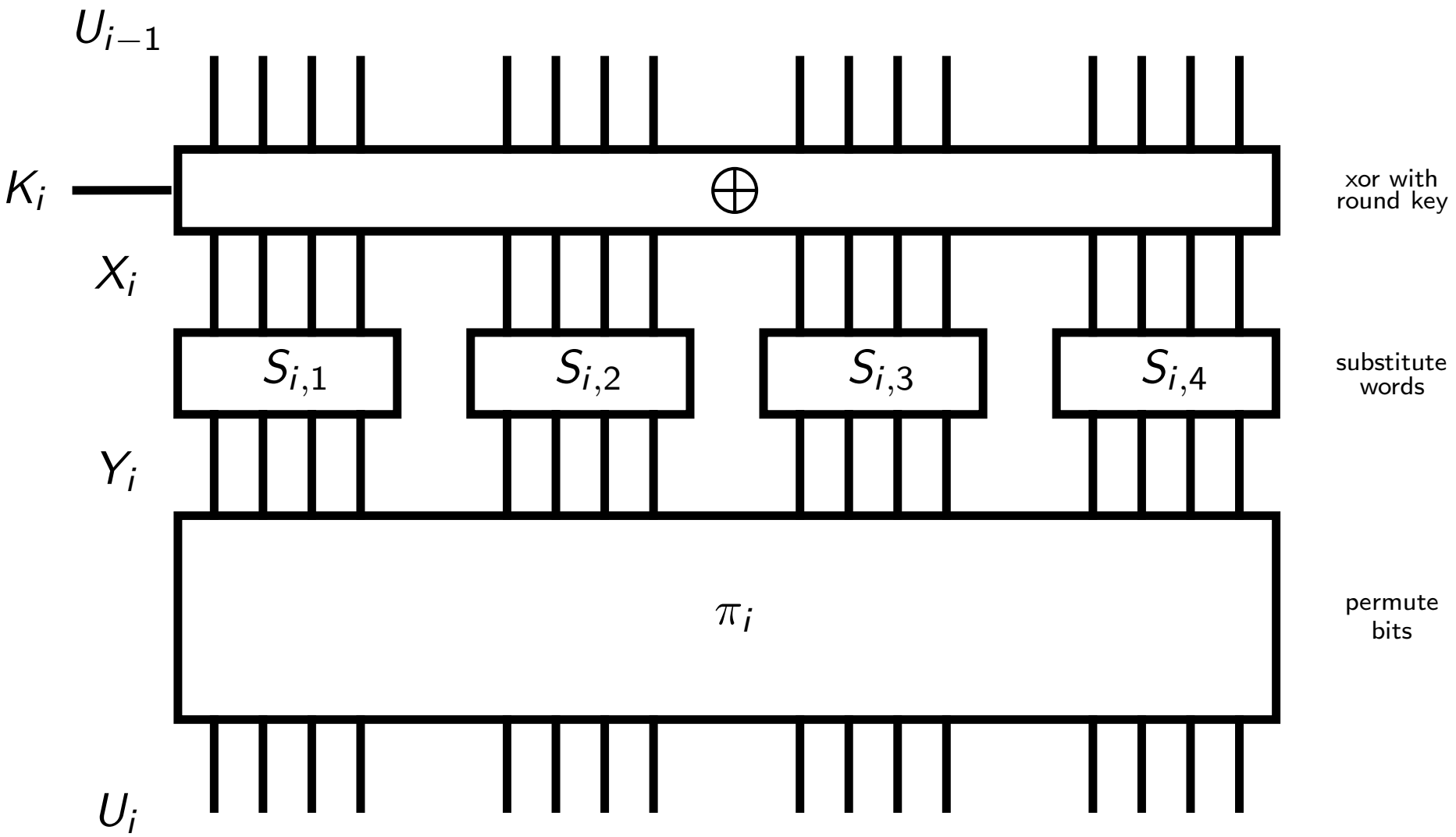
Substitution-Permutation Networks (2/2)



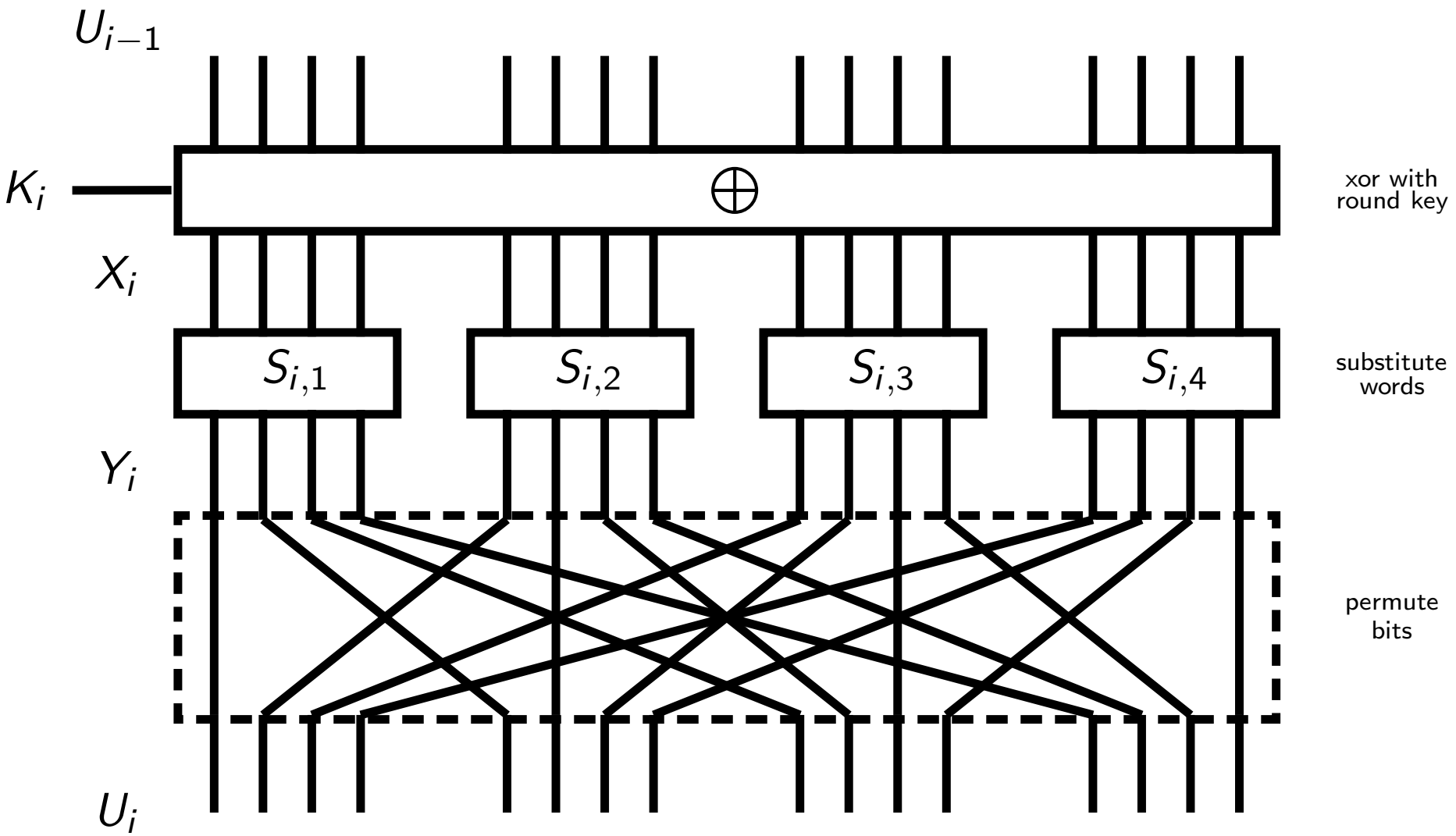
Substitution-Permutation Networks (2/2)



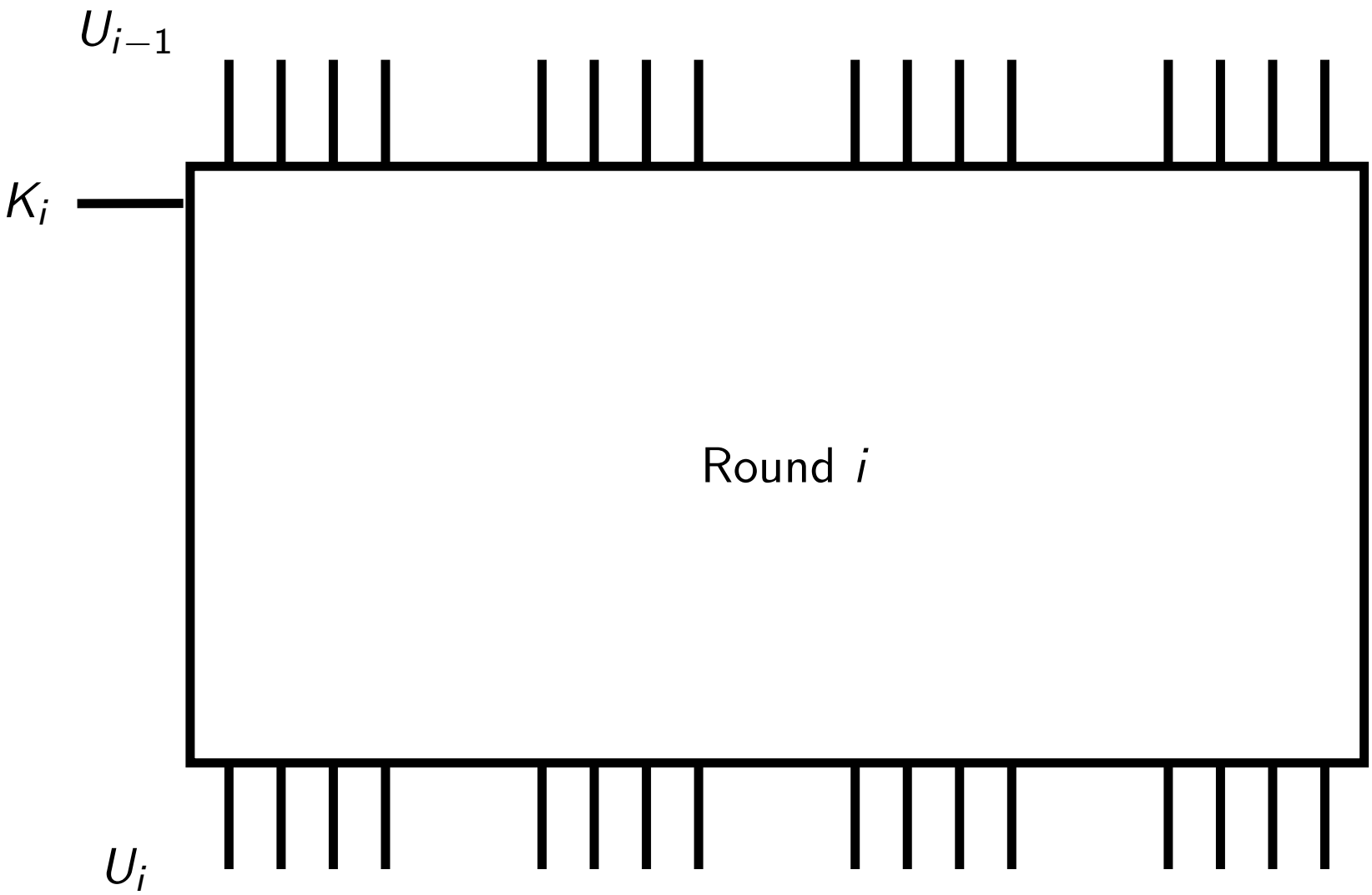
Substitution-Permutation Networks (2/2)



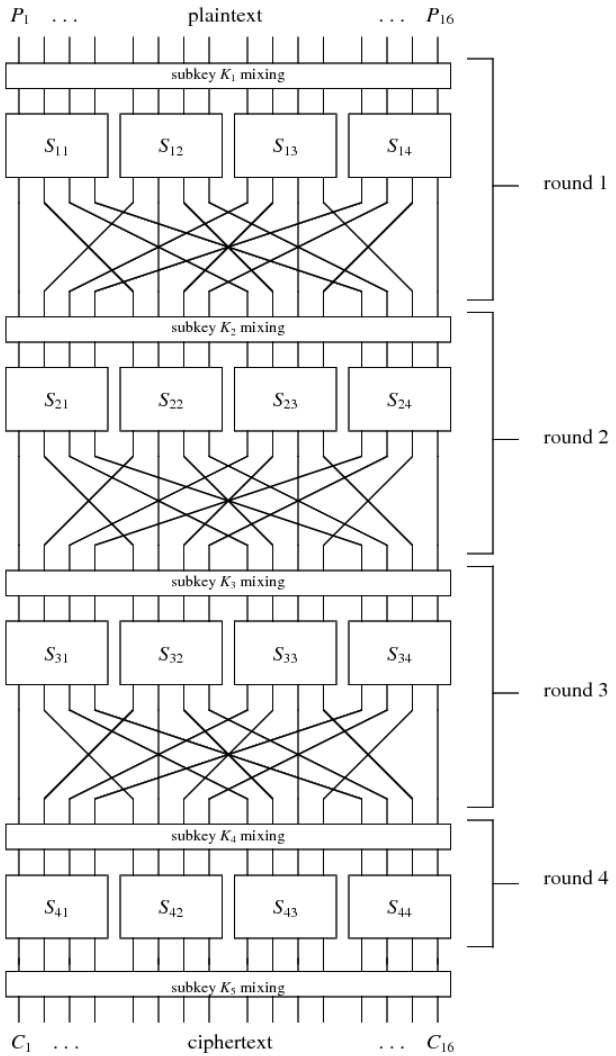
Substitution-Permutation Networks (2/2)



Substitution-Permutation Networks (2/2)



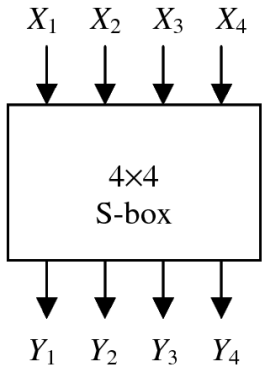
A Simple Block Cipher (1/2)



- $|P| = |C| = 16$
- 4 rounds
- $|K| = 32$
- rth round key K_r consists of the $4r$th to the $(4r + 16)$th bits of key K.
- 4-bit S-Boxes

A Simple Block Cipher (2/2)

S-Boxes the same ($S \neq S^{-1}$)

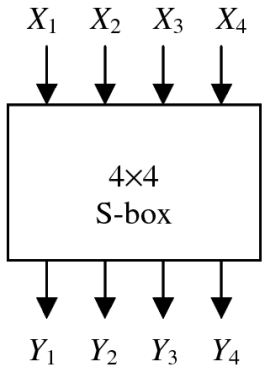


- $Y = S(X)$
- Can be described using 4 boolean functions

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

A Simple Block Cipher (2/2)

S-Boxes the same ($S \neq S^{-1}$)



- $Y = S(X)$
- Can be described using 4 boolean functions

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

16-bit permutation ($\pi = \pi^{-1}$)

Input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

AES

Advanced Encryption Standard (AES)

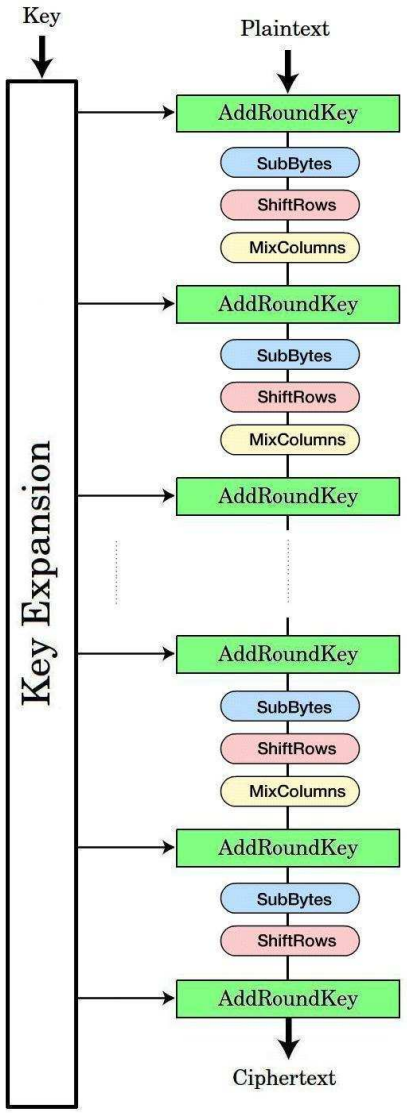
- Chosen in worldwide **public competition** 1997-2000. Probably no backdoors. Increased confidence!
- Winning proposal named "Rijndael", by Rijmen and Daemen
- Family of 128-bit block ciphers:

Key bits	128	192	256
Rounds	10	12	14

- The first key-recovery attacks on full AES due to Bogdanov, Khovratovich, and Rechberger, published **2011**, is faster than brute force by a factor of about **4**.
- ... algebraics of AES make some people uneasy.

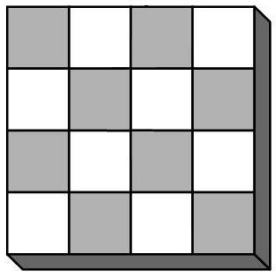
AES

- **AddRoundKey**: xor with round key.
- **SubBytes**: substitution of bytes.
- **ShiftRows**: permutation of bytes.
- **MixColumns**: linear map.



Similar to SPN

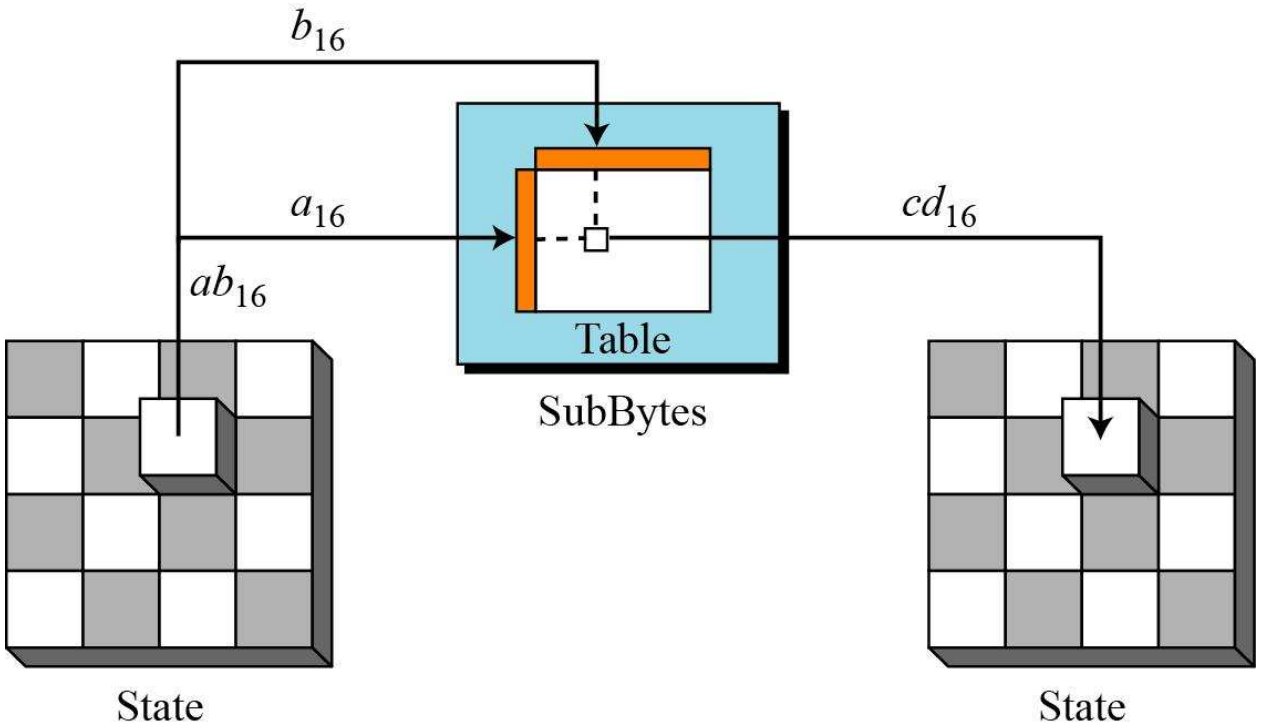
The 128 bit state is interpreted as a 4×4 matrix of bytes.



Something like a mix between substitution, permutation, affine version of Hill cipher. In each round!

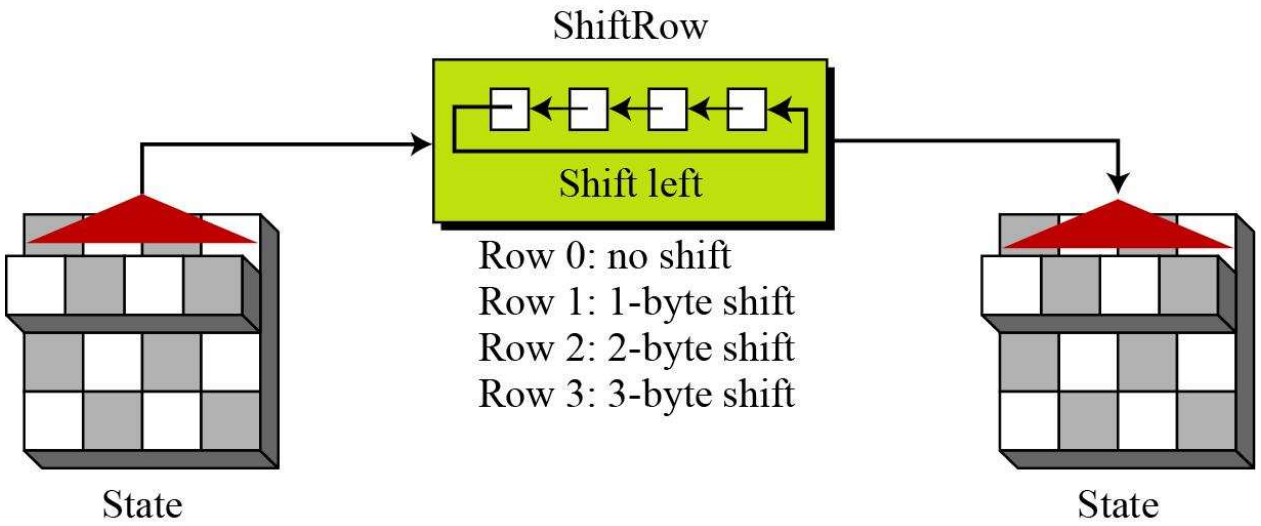
SubBytes

`SubBytes` is field inversion in $\mathbb{F}_{2^8}$ plus affine map in $\mathbb{F}_2^8$.



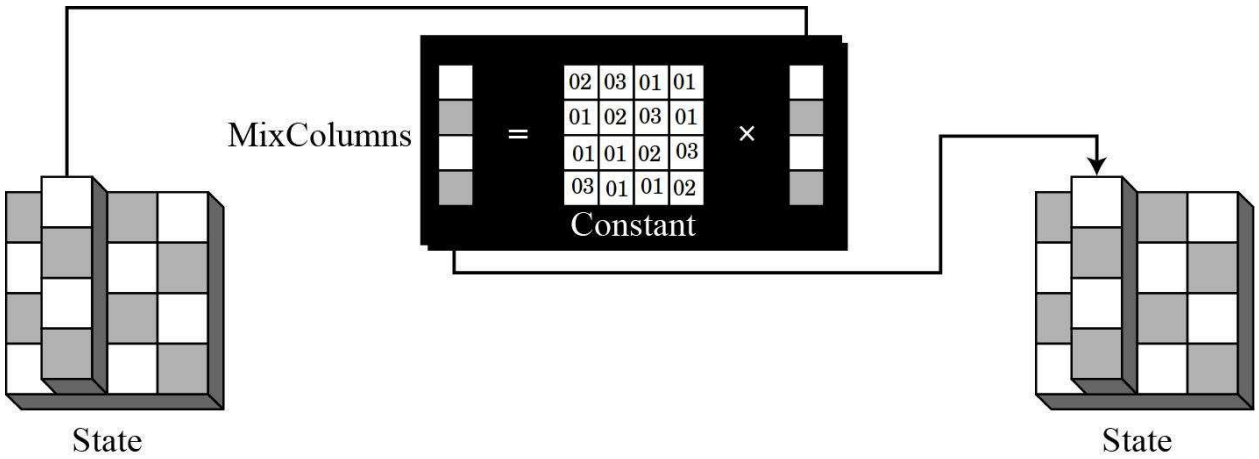
ShiftRows

`ShiftRows` is a cyclic shift of bytes with offsets: 0, 1, 2, and 3.



MixColumns

`MixColumns` is an invertible linear map over $\mathbb{F}_{2^8}$ (with irreducibile polynomial $x^8 + x^4 + x^3 + x + 1$) with good diffusion.



Decryption

Uses the following transforms:

- **AddRoundKey**
- **InvSubBytes**
- **InvShiftRows**
- **InvMixColumns**

Feistel Networks

Feistel Networks

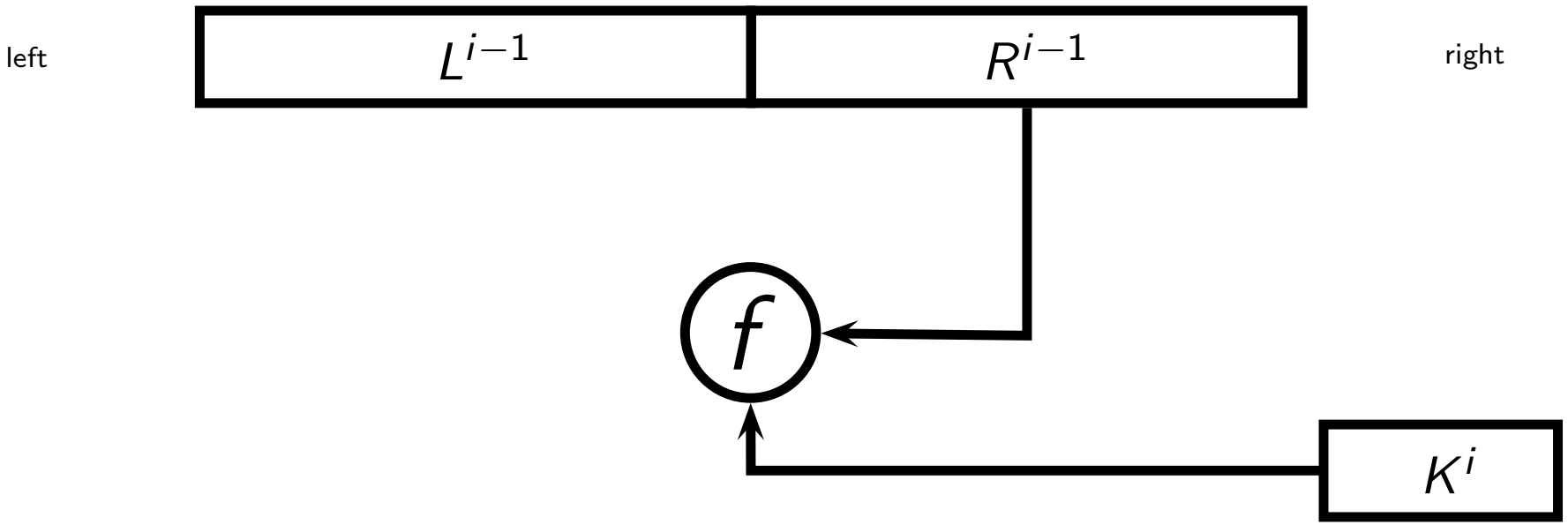
- Identical rounds are iterated, but with different round keys.
- The input to the ith round is divided in a left and right part, denoted L^{i-1} and R^{i-1}.
- f is a function for which it is somewhat hard to find pre-images, but f is typically **not invertible**!
- One round is defined by:

$$L^i = R^{i-1}$$
$$R^i = L^{i-1} \oplus f(R^{i-1}, K^i)$$

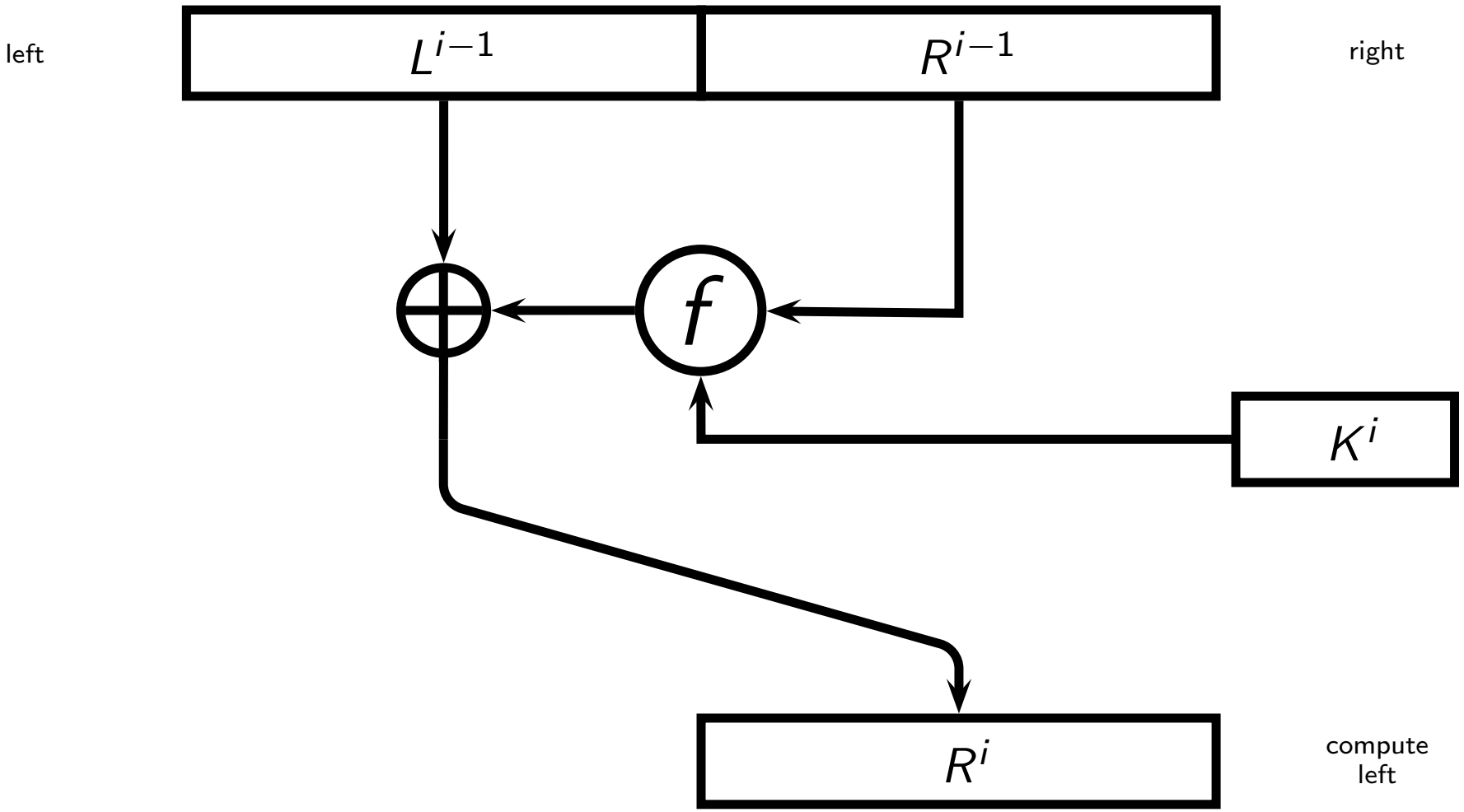
where K^i is the ith round key.

Feistel Round

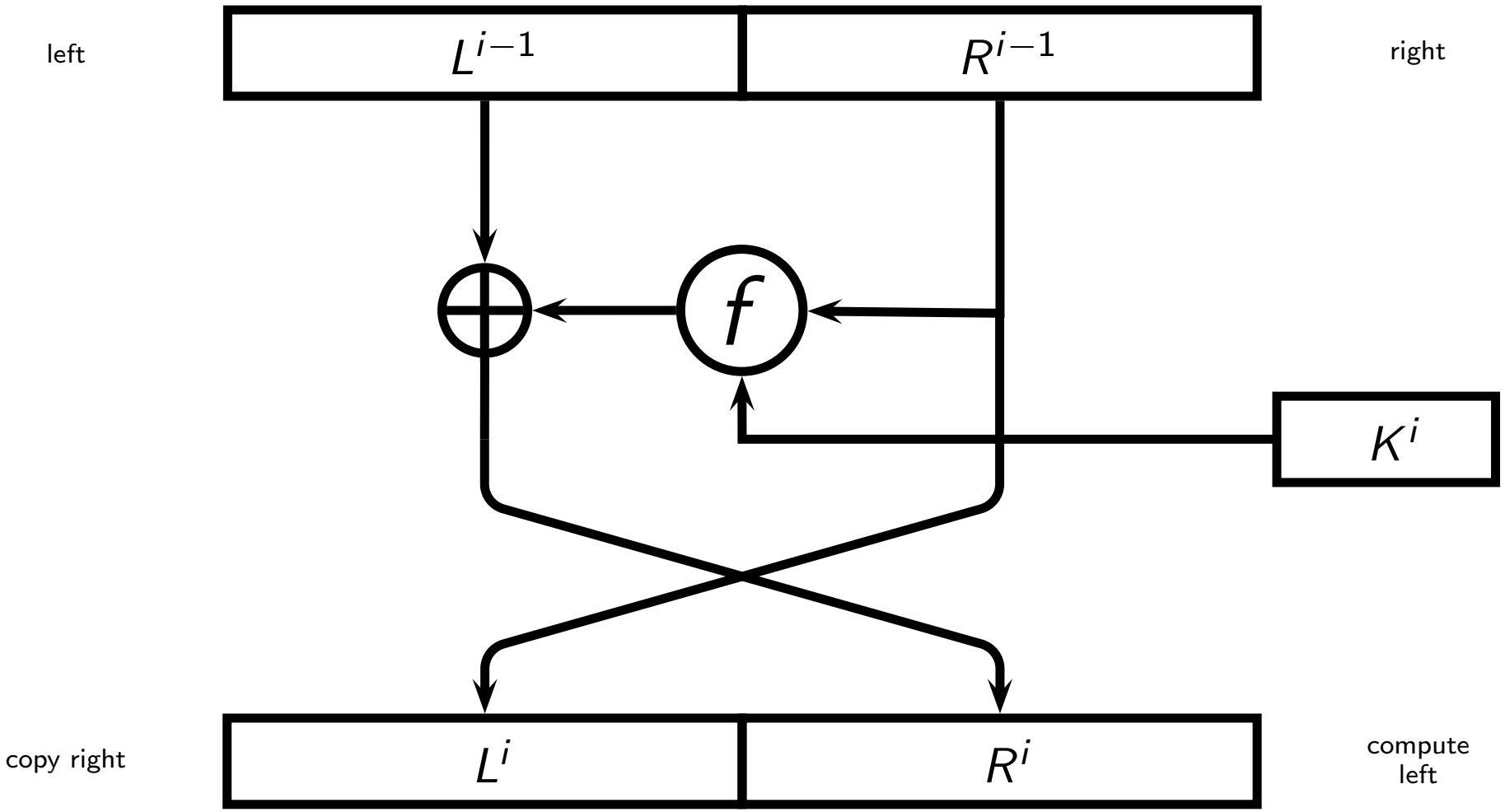
Feistel Round



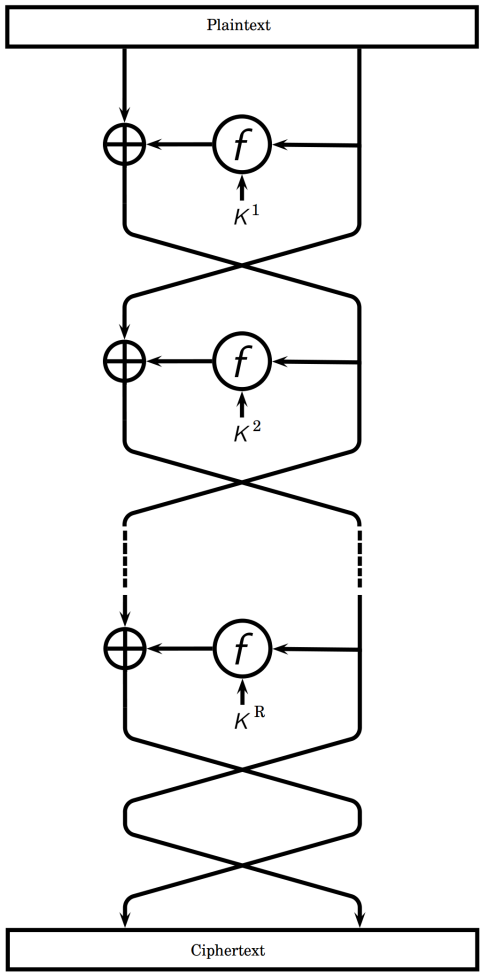
Feistel Round



Feistel Round



Feistel Cipher



Inverse Feistel Round

Feistel Round.

$$L^i = R^{i-1}$$
$$R^i = L^{i-1} \oplus f(R^{i-1}, K^i)$$

Inverse Feistel Round

Feistel Round.

$$L^i = R^{i-1}$$
$$R^i = L^{i-1} \oplus f(R^{i-1}, K^i)$$

Inverse Feistel Round.

$$L^{i-1} = R^i \oplus f(L^i, K^i)$$
$$R^{i-1} = L^i$$

Reverse direction and swap left and right!

DES

Quote

The news here is not that DES is insecure, that hardware algorithm-crackers can be built, or that a 56-bit key length is too short. ... The news is how long the government has been denying that these machines were possible. As recently as 8 June 98, Robert Litt, principal associate deputy attorney general at the Department of Justice, denied that it was possible for the FBI to crack DES. ... My comment was that the FBI is either incompetent or lying, or both.

– Bruce Schneier, 1998

Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...

Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.

Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.
- 16-round Feistel network.

Data Encryption Standard (DES)

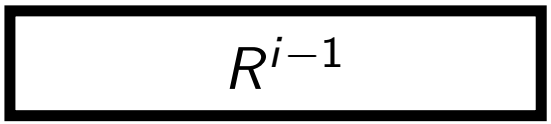
- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.
- 16-round Feistel network.
- Key schedule derives permuted bits for each round key from a 56-bit key. Supposedly not 64-bit due to parity bits.

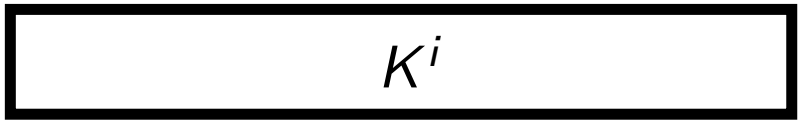
Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.
- 16-round Feistel network.
- Key schedule derives permuted bits for each round key from a 56-bit key. Supposedly not 64-bit due to parity bits.
- Let us look a little at the Feistel-function f.

DES's f-Function

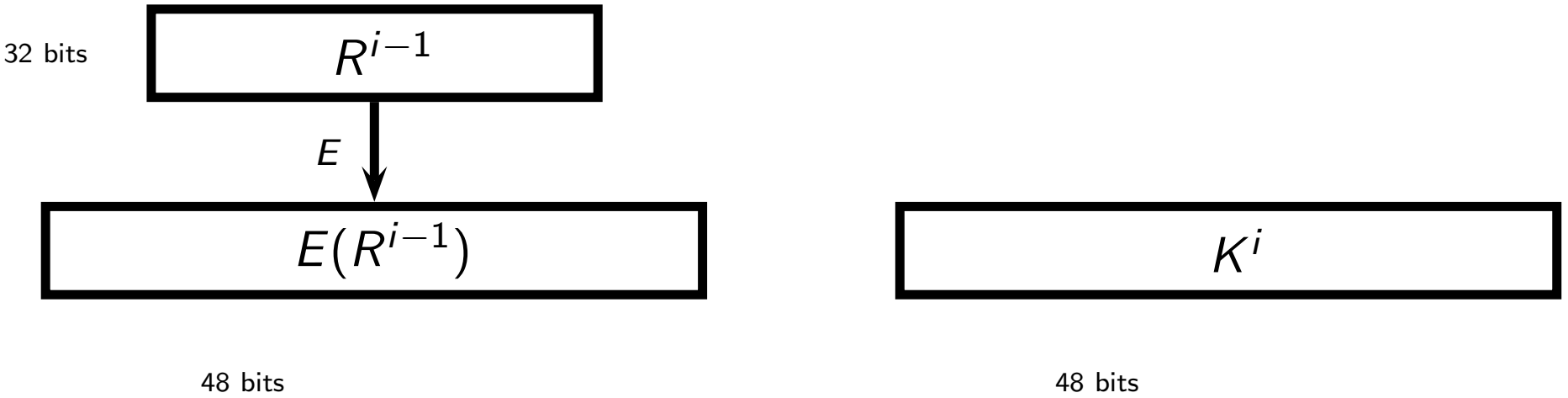
32 bits



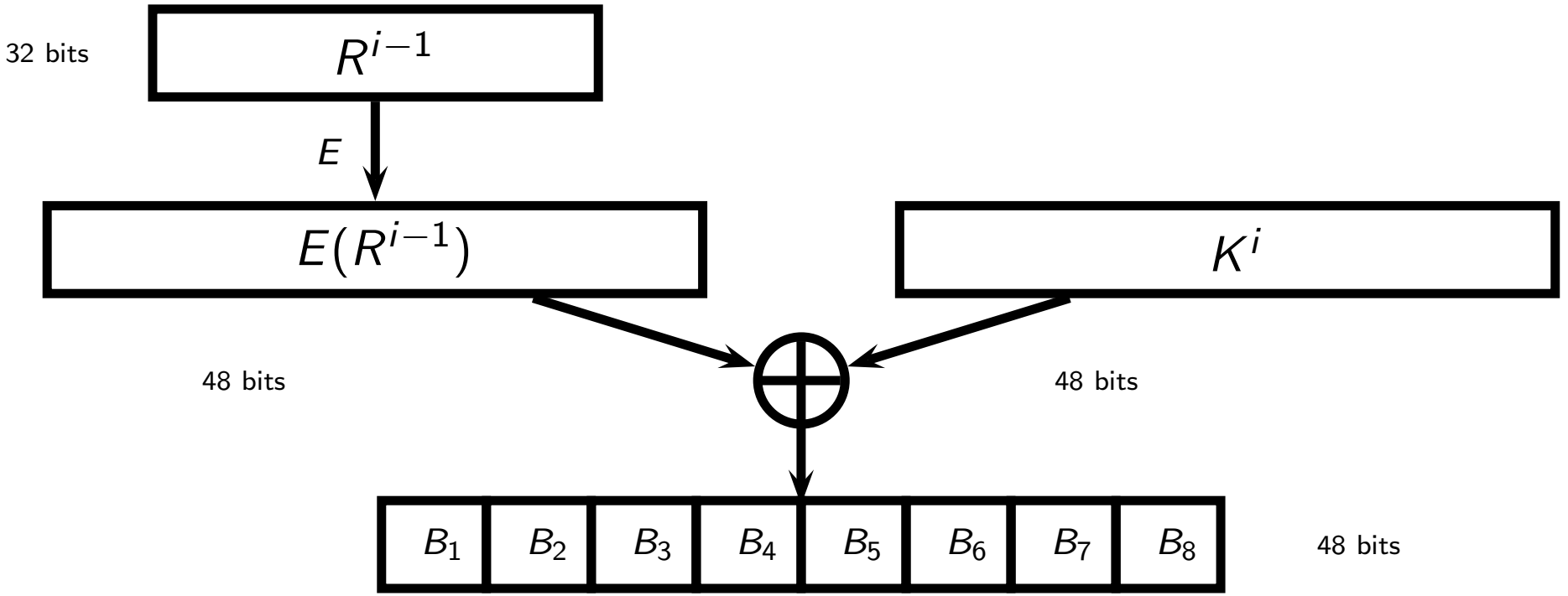


48 bits

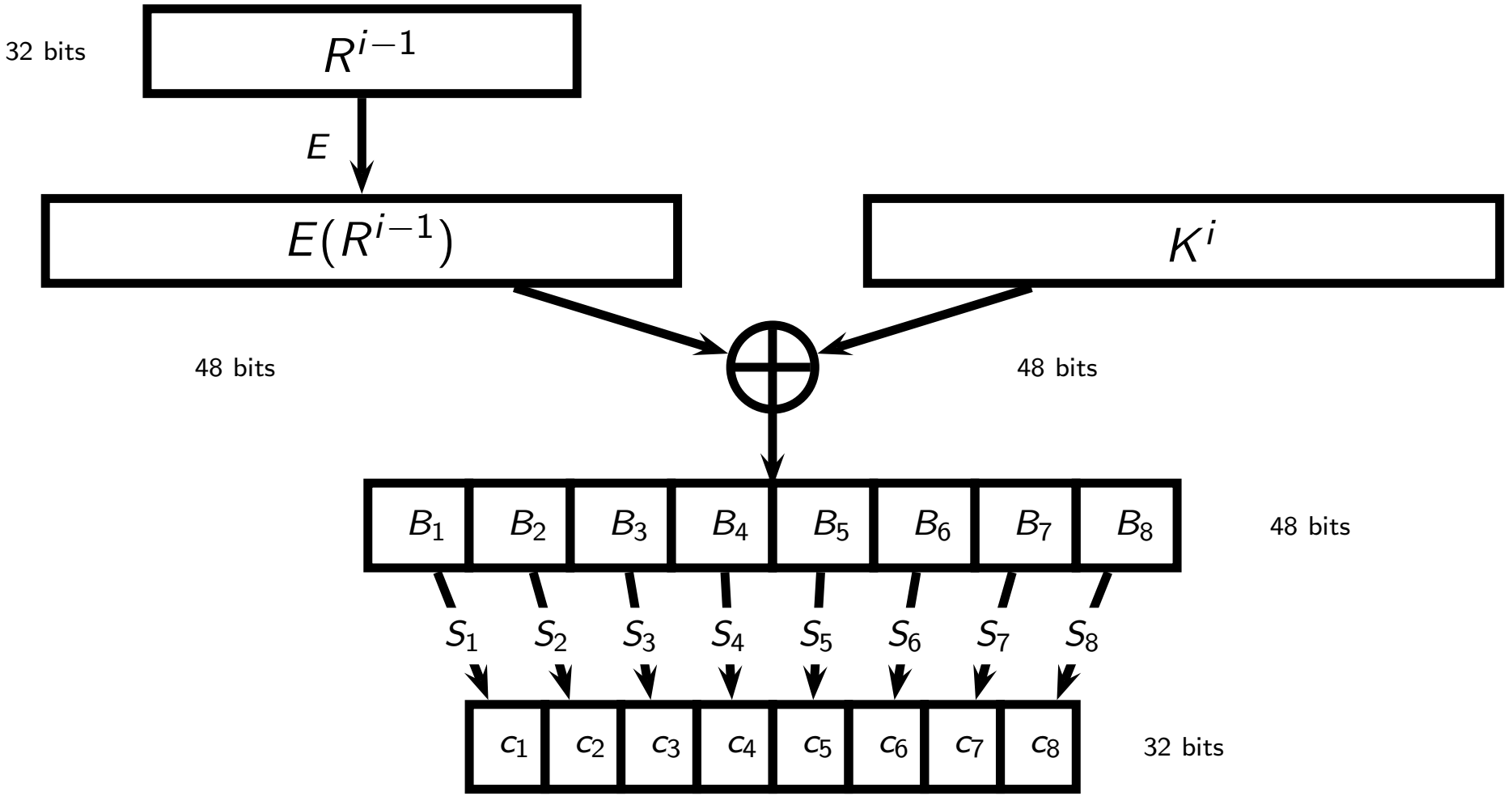
DES's f-Function



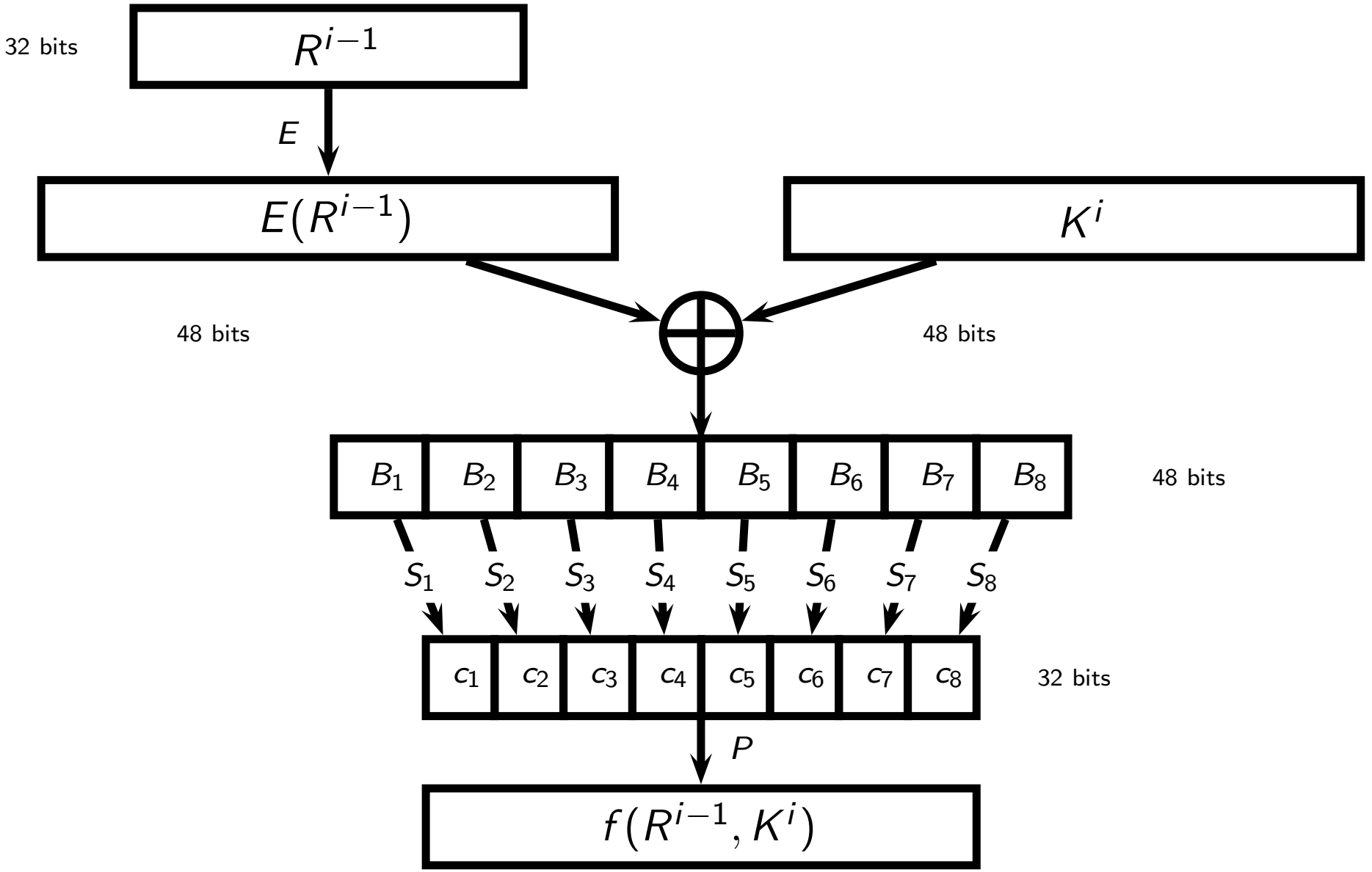
DES's f-Function



DES's f-Function



DES's f-Function



Security of DES

- **Brute Force.** Try all 2^{56} keys. Done in practice with special chip by Electronic Frontier Foundation, 1998. Likely much earlier by NSA and others.
- **Differential Cryptanalysis.** 2^{47} chosen plaintexts, Biham and Shamir, 1991. (approach: late 80'ies). Known earlier by IBM and NSA. DES is surprisingly resistant!
- **Linear Cryptanalysis.** 2^{43} known plaintexts, Matsui, 1993. Probably **not** known by IBM and NSA!

Double DES

We have seen that the key space of DES is too small. One way to increase it is to use DES twice, so called "double DES".

$$2\mathrm{DES}_{k_1,k_2}(x) = \mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(x))$$

Is this more secure than DES?

This question is valid for any cipher.

Meet-In-the-Middle Attack

- Get hold of a plaintext-ciphertext pair (m, c)
- Compute $X = \{x \mid k_1 \in \mathcal{K}_{\mathrm{DES}} \wedge x = \mathrm{E}_{k_1}(m)\}$.
- For $k_2 \in \mathcal{K}_{\mathrm{DES}}$ check if $\mathrm{E}_{k_2}^{-1}(c) = \mathrm{E}_{k_1}(m)$ for some k_1 using the table X. If so, then (k_1, k_2) is a good candidate.
- Repeat with (m', c'), starting from the set of candidate keys to identify correct key.

Triple DES

What about triple DES?

$$3\mathrm{DES}_{k_1,k_2,k_3}(x) = \mathrm{DES}_{k_3}(\mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(x)))$$

- Seemingly 112 bit "effective" key size.
- 3 times as slow as DES. DES is slow in software, and this is even worse. One of the motivations of AES.
- Triple DES is still considered to be secure.

Modes of Operation

Modes of Operation

- Electronic codebook mode (ECB mode).
- Cipher feedback mode (CFB mode).
- Cipher block chaining mode (CBC mode).
- Output feedback mode (OFB mode).
- Counter mode (CTR mode).

ECB Mode

Electronic codebook mode

Encrypt each block independently:

$$c_i = \mathrm{E}_k(m_i)$$

ECB Mode

Electronic codebook mode

Encrypt each block independently:

$$c_i = \mathrm{E}_k(m_i)$$

- Identical plaintext blocks give identical ciphertext blocks.

ECB Mode

Electronic codebook mode

Encrypt each block independently:

$$c_i = \mathrm{E}_k(m_i)$$

- Identical plaintext blocks give identical ciphertext blocks.
- How can we avoid this?

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

- Sequential encryption and parallel decryption.

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

- Sequential encryption and parallel decryption.
- Self-synchronizing.

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

- Sequential encryption and parallel decryption.
- Self-synchronizing.
- How do we pick the initialization vector?

CBC Mode

Cipher block chaining mode

xor plaintext block with previous ciphertext block **before** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = \mathsf{E}_k\left(c_{i-1} \oplus m_i\right)$$

CBC Mode

Cipher block chaining mode

xor plaintext block with previous ciphertext block **before** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = \mathsf{E}_k\left(c_{i-1} \oplus m_i\right)$$

- Sequential encryption and parallel decryption

CBC Mode

Cipher block chaining mode

xor plaintext block with previous ciphertext block **before** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = \mathsf{E}_k\left(c_{i-1} \oplus m_i\right)$$

- Sequential encryption and parallel decryption
- Self-synchronizing.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$s_0 = \text{initialization vector}$$
$$s_i = \mathsf{E}_k(s_{i-1})$$
$$c_i = s_i \oplus m_i$$

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Sequential.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Sequential.
- Synchronous.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Sequential.
- Synchronous.
- Allows batch processing.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Sequential.
- Synchronous.
- Allows batch processing.
- Malleable!

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Parallel.

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Parallel.
- Synchronous.

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Parallel.
- Synchronous.
- Allows batch processing.

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Parallel.
- Synchronous.
- Allows batch processing.
- Malleable!

Linear Cryptanalysis of the SPN

Basic Idea – Linearize

Find an expression of the following form with a high probability of occurrence.

$$P_{i_1} \oplus \cdots \oplus P_{i_p} \oplus C_{j_1} \oplus \cdots \oplus C_{j_c} = K_{\ell_1,s_1} \oplus \cdots \oplus K_{\ell_k,s_k}$$

Each random plaintext/ciphertext pair gives an estimate of

$$K_{\ell_1,s_1} \oplus \cdots \oplus K_{\ell_k,s_k}$$

Collect many pairs and make a better estimate based on the majority vote.

How do we come up with the desired expression?

How do we compute the required number of samples?

Bias

Definition. The bias $\epsilon(X)$ of a binary random variable X is defined by

$$\epsilon(X) = \Pr[X = 0] - \frac{1}{2} \ .$$

Bias

Definition. The bias $\epsilon(X)$ of a binary random variable X is defined by

$$\epsilon(X) = \Pr[X = 0] - \frac{1}{2} \ .$$

$\approx 1/\epsilon^2(X)$ samples are required to estimate X (Matsui)

Linear Approximation of S-Box (1/3)

Let X and Y be the input and output of an S-box, i.e.

$$Y = S(X) \ .$$

We consider the bias of linear combinations of the form

$$a \cdot X \oplus b \cdot Y = \left(\bigoplus_i a_i X_i \right) \oplus \left(\bigoplus_i b_i Y_i \right) \ .$$

Linear Approximation of S-Box (1/3)

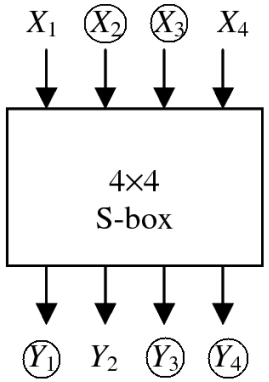
Let X and Y be the input and output of an S-box, i.e.

$$Y = S(X) \ .$$

We consider the bias of linear combinations of the form

$$a \cdot X \oplus b \cdot Y = \left(\bigoplus_i a_i X_i \right) \oplus \left(\bigoplus_i b_i Y_i \right) \ .$$

Example: $X_2 \oplus X_3 = Y_1 \oplus Y_3 \oplus Y_4$
The expression holds in 12 out of the 16 cases. Hence, it has a bias of $(12 - 8)/16 = 4/16 = 1/4$.



Linear Approximation of S-Box (2/3)

- Let $N_L(a, b)$ be the number of zero-outcomes of $a \cdot X \oplus b \cdot Y$.
- The bias is then

$$\epsilon(a \cdot X \oplus b \cdot Y) = \frac{N_L(a, b) - 8}{16} \quad ,$$

since there are four bits in X, and Y is determined by X.

Linear Approximation Table (3/3)

$$N_L(a, b) - 8$$

Input Sum	Output Sum 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	−2	−2	0	0	−2	+6	+2	+2	0	0	+2	+2	0	0
2	0	0	−2	−2	0	0	−2	−2	0	0	+2	+2	0	0	−6	+2
3	0	0	0	0	0	0	0	0	+2	−6	−2	−2	+2	+2	−2	−2
4	0	+2	0	−2	−2	−4	−2	0	0	−2	0	+2	+2	−4	+2	0
5	0	−2	−2	0	−2	0	+4	+2	−2	0	−4	+2	0	−2	−2	0
6	0	+2	−2	+4	+2	0	0	+2	0	−2	+2	+4	−2	0	0	−2
7	0	−2	0	+2	+2	−4	+2	0	−2	0	+2	0	+4	+2	0	+2
8	0	0	0	0	0	0	0	0	−2	+2	+2	−2	+2	−2	−2	−6
9	0	0	−2	−2	0	0	−2	−2	−4	0	−2	+2	0	+4	+2	−2
A	0	+4	−2	+2	−4	0	+2	−2	+2	+2	0	0	+2	+2	0	0
B	0	+4	0	−4	+4	0	+4	0	0	0	0	0	0	0	0	0
C	0	−2	+4	−2	−2	0	+2	0	+2	0	+2	+4	0	+2	0	−2
D	0	+2	+2	0	−2	+4	0	+2	−4	−2	+2	0	+2	0	0	+2
E	0	+2	+2	0	−2	−4	0	+2	−2	0	0	−2	−4	+2	−2	0
F	0	−2	−4	−2	−2	0	+2	0	0	−2	+4	−2	−2	0	+2	0

This gives linear approximation for one round.

How do we come up with linear approximation for more rounds?

Piling-Up Lemma

Lemma. Let $X_1, \ldots, X_t$ be independent binary random variables and let $\epsilon_i = \epsilon(X_i)$. Then

$$\epsilon\left(\bigoplus_i X_i\right) = 2^{t-1}\prod_i \epsilon_i \ .$$

Proof. Case $t = 2$:

$$\begin{aligned}\Pr[X_1 \oplus X_2 = 0] &= \Pr[(X_1 = 0 \wedge X_1 = 0) \vee (X_1 = 1 \wedge X_1 = 1)] \\ &= (\frac{1}{2} + \epsilon_1)(\frac{1}{2} + \epsilon_2) + (\frac{1}{2} - \epsilon_1)(\frac{1}{2} - \epsilon_2) \\ &= \frac{1}{2} + 2\epsilon_1\epsilon_2 \ .\end{aligned}$$

By induction $\Pr[X_1 \oplus \cdots \oplus X_t = 0] = \frac{1}{2} + 2^{t-1}\prod_i \epsilon_i$

Linear Trail

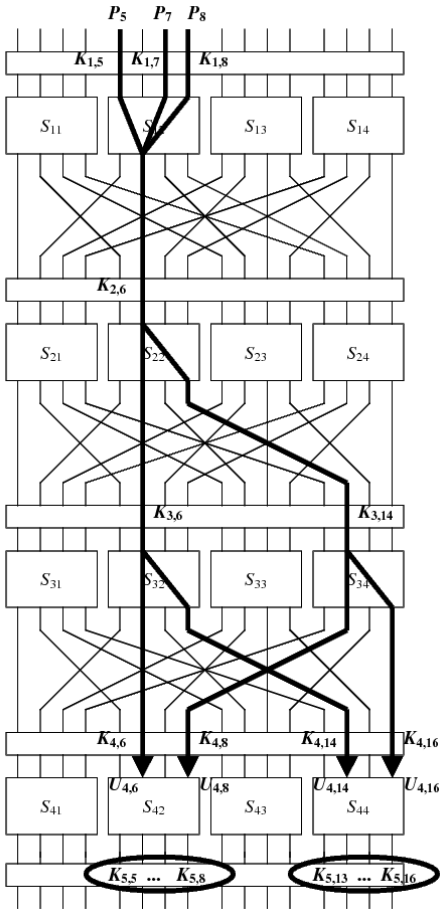
Four linear approximations with $|\epsilon_i| = 1/4$

$$
\begin{aligned}
S_{12}: &\quad X_1 \oplus X_3 \oplus X_4 = Y_2 \\
S_{22}: &\quad X_2 = Y_2 \oplus Y_4 \\
S_{32}: &\quad X_2 = Y_2 \oplus Y_4 \\
S_{34}: &\quad X_2 = Y_2 \oplus Y_4
\end{aligned}
$$

Combine them to get:

$$U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,16} \oplus P_5 \oplus P_7 \oplus P_8 = \bigoplus K_{i,j}$$

with bias $|\epsilon| = 2^{4-1}(\frac{1}{4})^4 = 2^{-5}$



Attack Idea

- Our expression (with bias 2^{-5}) links plaintext bits to input bits to the 4th round
- Partially undo the last round by guessing the last key. Only 2 S-Boxes are involved, i.e., $2^8 = 256$ guesses
- For a correct guess, the equation holds with bias 2^{-5}. For a wrong guess, it holds with bias zero (i.e., probability close to $1/2$).

Attack Idea

- Our expression (with bias 2^{-5}) links plaintext bits to input bits to the 4th round
- Partially undo the last round by guessing the last key. Only 2 S-Boxes are involved, i.e., $2^8 = 256$ guesses
- For a correct guess, the equation holds with bias 2^{-5}. For a wrong guess, it holds with bias zero (i.e., probability close to 1/2).

Required pairs $2^{10} \approx 1000$
Attack complexity $2^{18} \ll 2^{32}$ operations

Linear Cryptanalysis Summary

1. Find linear approximation of S-Boxes.
2. Compute bias of each approximation.
3. Find linear trails.
4. Compute bias of linear trails.
5. Compute data and time complexity.
6. Estimate key bits from many plaintext-ciphertexts pairs.

Linear cryptanalysis is a **known plaintext attack**.

Ideal Block Cipher

Negligible Functions

Definition. A function $\epsilon(n)$ is negligible if for every constant $c > 0$, there exists a constant n_0, such that

$$\epsilon(n) < \frac{1}{n^c}$$

for all $n \geq n_0$.

Motivation. Events happening with negligible probability can not be exploited by polynomial time algorithms! (they "never" happen)

Pseudo-Random Function

"Definition". A function is pseudo-random if no efficient adversary can distinguish between the function and a random function.

Pseudo-Random Function

"Definition". A function is pseudo-random if no efficient adversary can distinguish between the function and a random function.

Definition. A family of functions $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is pseudo-random if for all polynomial time oracle adversaries A

$$\left| \Pr_K \left[A^{F_K(\cdot)} = 1 \right] - \Pr_{R:\{0,1\}^n \to \{0,1\}^n} \left[A^{R(\cdot)} = 1 \right] \right|$$

is negligible.

Pseudo-Random Permutation

"Definition". A permutation and its inverse is pseudo-random if no efficient adversary can distinguish between the permutation and its inverse, and a random permutation and its inverse.

Pseudo-Random Permutation

"Definition". A permutation and its inverse is pseudo-random if no efficient adversary can distinguish between the permutation and its inverse, and a random permutation and its inverse.

Definition. A family of permutations $P : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ are pseudo-random if for all polynomial time oracle adversaries A

$$\left| \Pr_K \left[A^{P_K(\cdot), P_K^{-1}(\cdot)} = 1 \right] - \Pr_{\Pi \in \mathcal{S}_{2^n}} \left[A^{\Pi(\cdot), \Pi^{-1}(\cdot)} = 1 \right] \right|$$

is negligible, where $\mathcal{S}_{2^n}$ is the set of permutations of $\{0,1\}^n$.

Idealized Four-Round Feistel Network

Definition. Feistel round (H for "Horst Feistel").

$$H_{F_K}(L, R) = (R, L \oplus F(R, K))$$

Idealized Four-Round Feistel Network

Definition. Feistel round (H for "Horst Feistel").

$$H_{F_K}(L, R) = (R, L \oplus F(R, K))$$

Theorem. (Luby and Rackoff) If F is a pseudo-random family of functions, then

$$H_{F_{k_1}, F_{k_2}, F_{k_3}, F_{k_4}}(x) = H_{F_{k_4}}(H_{F_{k_3}}(H_{F_{k_2}}(H_{F_{k_1}}(x))))$$

(and its inverse) is a pseudo-random family of permutations.

Idealized Four-Round Feistel Network

Definition. Feistel round (H for "Horst Feistel").

$$H_{F_K}(L, R) = (R, L \oplus F(R, K))$$

Theorem. (Luby and Rackoff) If F is a pseudo-random family of functions, then

$$H_{F_{k_1}, F_{k_2}, F_{k_3}, F_{k_4}}(x) = H_{F_{k_4}}(H_{F_{k_3}}(H_{F_{k_2}}(H_{F_{k_1}}(x))))$$

(and its inverse) is a pseudo-random family of permutations.

Why do we need four rounds?

Perfect Secrecy

Perfect Secrecy (1/3)

When is a cipher perfectly secure?

Perfect Secrecy (1/3)

When is a cipher perfectly secure?

How should we formalize this?

Perfect Secrecy (2/3)

Definition. A cryptosystem has perfect secrecy if guessing the plaintext is as hard to do given the ciphertext as it is without it.

Perfect Secrecy (2/3)

Definition. A cryptosystem has perfect secrecy if guessing the plaintext is as hard to do given the ciphertext as it is without it.

Definition. A cryptosystem has perfect secrecy if

$$\Pr[M = m \,|\, C = c] = \Pr[M = m]$$

for every $m \in \mathcal{M}$ and $c \in \mathcal{C}$, where M and C are random variables taking values over $\mathcal{M}$ and $\mathcal{C}$.

Perfect Secrecy (3/3)

Game Based Definition. Exp_A^b, where A is a strategy:

1. $\mathsf{k} \leftarrow_R \mathcal{K}$
2. $(m_0, m_1) \leftarrow A$
3. $c = \mathsf{E}_{\mathsf{k}}(m_b)$
4. $d \leftarrow A(c)$, with $d \in \{0, 1\}$
5. Output d.

Definition. A cryptosystem has perfect secrecy if for every **computationally unbounded** strategy A,

$$\Pr\left[\mathrm{Exp}_A^0 = 1\right] = \Pr\left[\mathrm{Exp}_A^1 = 1\right] \quad .$$

One-Time Pad

One-Time Pad (OTP).

- **Key.** Random tuple $\mathrm{k} = (b_0, \ldots, b_{n-1}) \in \mathbb{Z}_2^n$.
- **Encrypt.** Plaintext $m = (m_0, \ldots, m_{n-1}) \in \mathbb{Z}_2^n$ gives ciphertext $c = (c_0, \ldots, c_{n-1})$, where $c_i = m_i \oplus b_i$.
- **Decrypt.** Ciphertext $c = (c_0, \ldots, c_{n-1}) \in \mathbb{Z}_2^n$ gives plaintext $m = (m_0, \ldots, m_{n-1})$, where $m_i = c_i \oplus b_i$.

Bayes' Theorem

Theorem. If A and B are events and $\Pr[B] > 0$, then

$$\Pr[A \mid B] = \frac{\Pr[A] \Pr[B \mid A]}{\Pr[B]}$$

Terminology:

$\Pr[A]$ – prior probability of A
$\Pr[B]$ – prior probability of B
$\Pr[A \mid B]$ – posterior probability of A given B
$\Pr[B \mid A]$ – posterior probability of B given A

One-Time Pad Has Perfect Secrecy

- **Probabilistic Argument.** Bayes implies that:

$$\begin{aligned}\Pr[M = m \,|\, C = c] &= \frac{\Pr[M = m]\Pr[C = c \,|\, M = m]}{\Pr[C = c]} \\ &= \Pr[M = m]\,\frac{2^{-n}}{2^{-n}} \\ &= \Pr[M = m] \quad .\end{aligned}$$

- **Simulation Argument.** The ciphertext is uniformly and independently distributed from the plaintext. We can **simulate** it on our own!

Bad News

Theorem. "For every cipher with perfect secrecy, the key requires at least as much space to represent as the plaintext."

Dangerous in practice to rely on no reuse of, e.g., file containing randomness!

Information Theory

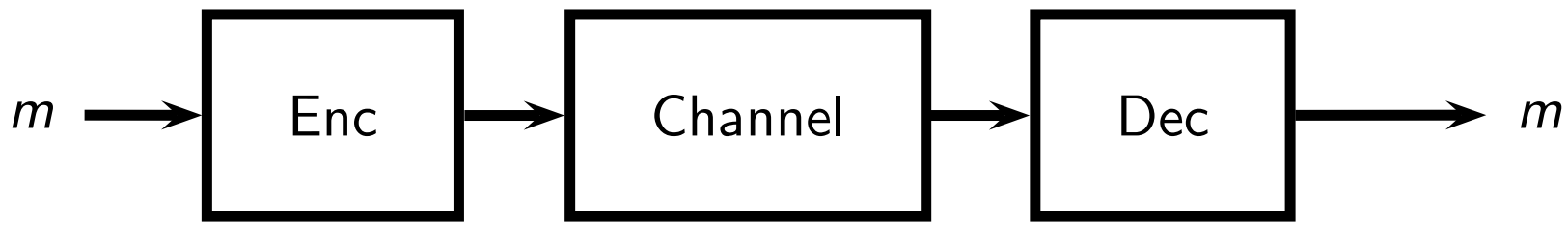
Information Theory

- Information theory is a mathematical theory of communication.
- Typical questions studied are how to compress, transmit, and store information.
- Information theory is also useful to argue about some cryptographic schemes and protocols.

Classical Information Theory

- **Memoryless Source Over Finite Alphabet.** A source produces symbols from an alphabet $\Sigma = \{a_1, \ldots, a_n\}$. Each generated symbol is independently distributed.
- **Binary Channel.** A binary channel can (only) send bits.
- **Coder/Decoder.** Our goal is to come up with a scheme to:
  1. convert a symbol a from the alphabet Σ into a sequence $(b_1, \ldots, b_l)$ of bits,
  2. send the bits over the channel, and
  3. decode the sequence into a again at the receiving end.

Classical Information Theory



Alice

Bob

Optimization Goal

We want to minimize the **expected** number of bits/symbol we send over the binary channel, i.e., if X is a random variable over Σ and $l(x)$ is the length of the codeword of x then we wish to minimize

$$\mathrm{E}\left[l(X)\right] = \sum_{x \in \Sigma} \mathrm{P}_X\left(x\right) l(x) \ .$$

Examples:

- X takes values in $\Sigma = \{a, b, c, d\}$ with uniform distribution. How would you encode this?

Examples:

- X takes values in $\Sigma = \{a, b, c, d\}$ with uniform distribution. How would you encode this?

It seems we need $I(x) = \log |\Sigma|$. This gives the Hartley measure.

Examples:

- X takes values in $\Sigma = \{a, b, c, d\}$ with uniform distribution. How would you encode this?
- X takes values in $\Sigma = \{a, b, c\}$, with $\mathrm{P}_X(a) = \frac{1}{2}$, $\mathrm{P}_X(b) = \frac{1}{4}$, and $\mathrm{P}_X(c) = \frac{1}{4}$. How would you encode this?

It seems we need $I(x) = \log |\Sigma|$. This gives the Hartley measure. **hmmm...**

Examples:

- X takes values in $\Sigma = \{a, b, c, d\}$ with uniform distribution. How would you encode this?
- X takes values in $\Sigma = \{a, b, c\}$, with $\mathrm{P}_X(a) = \frac{1}{2}$, $\mathrm{P}_X(b) = \frac{1}{4}$, and $\mathrm{P}_X(c) = \frac{1}{4}$. How would you encode this?

It seems we need $I(x) = \log \frac{1}{\mathrm{P}_X(x)}$ bits to encode x.

Entropy

Let us turn this expression into a definition.

Definition. Let X be a random variable taking values in $\mathcal{X}$. Then the **entropy** of X is

$$H(X) = -\sum_{x \in \mathcal{X}} \mathrm{P}_X(x) \log \mathrm{P}_X(x) \quad .$$

Examples and intuition are nice, but what we need is a theorem that states that this is **exactly** the right expected length of an optimal code.

Jensen's Inequality

Definition. A function $f : \mathcal{X} \to (a, b)$ is **concave** if

$$\lambda \cdot f(x) + (1 - \lambda) f(y) \leq f\left(\lambda \cdot x + (1 - \lambda) y\right) \quad ,$$

for every $x, y \in (a, b)$ and $0 \leq \lambda \leq 1$.

Jensen's Inequality

Definition. A function $f : \mathcal{X} \to (a, b)$ is **concave** if

$$\lambda \cdot f(x) + (1 - \lambda) f(y) \leq f\left(\lambda \cdot x + (1 - \lambda) y\right) \quad ,$$

for every $x, y \in (a, b)$ and $0 \leq \lambda \leq 1$.

Theorem. Suppose f is continuous and strictly concave on (a, b), and X is a discrete random variable. Then

$$\mathrm{E}\left[f(X)\right] \leq f(\mathrm{E}\left[X\right]) \quad ,$$

with equality iff X is constant.

Jensen's Inequality

Definition. A function $f : \mathcal{X} \to (a, b)$ is **concave** if

$$\lambda \cdot f(x) + (1 - \lambda) f(y) \leq f\left(\lambda \cdot x + (1 - \lambda) y\right) \quad ,$$

for every $x, y \in (a, b)$ and $0 \leq \lambda \leq 1$.

Theorem. Suppose f is continuous and strictly concave on (a, b), and X is a discrete random variable. Then

$$\mathrm{E}\left[f(X)\right] \leq f(\mathrm{E}\left[X\right]) \quad ,$$

with equality iff X is constant.

Proof idea. Consider two points + induction over number of points.

Kraft's Inequality

Theorem. There exists a prefix-free code E with codeword lengths l_x, for $x \in \Sigma$ if and only if

$$\sum_{x \in \Sigma} 2^{-l_x} \leq 1 \ .$$

Proof Sketch. $\Rightarrow$ Given a prefix-free code, we consider the corresponding binary tree with codewords at the leaves. We may "fold" it by replacing two sibling leaves $\mathrm{E}(x)$ and $\mathrm{E}(y)$ by (xy) with length $l_x - 1$. Repeat.

$\Leftarrow$ Given lengths $l_{x_1} \leq l_{x_2} \leq \ldots \leq l_{x_n}$ we start with the complete binary tree of depth l_{x_n} and prune it.

Binary Source Coding Theorem (1/2)

Theorem. Let E be an optimal code and let $l(x)$ be the length of the codeword of x. Then

$$H(X) \leq \mathrm{E}\left[l(X)\right] < H(X) + 1 \ .$$

Binary Source Coding Theorem (1/2)

Theorem. Let E be an optimal code and let $l(x)$ be the length of the codeword of x. Then

$$H(X) \leq \mathrm{E}\left[l(X)\right] < H(X) + 1 \ .$$

Proof of Upper Bound.
Define $l_x = \lceil -\log \mathrm{P}_X(x) \rceil$. Then we have

$$\sum_{x \in \Sigma} 2^{-l_x} \leq \sum_{x \in \Sigma} 2^{\log \mathrm{P}_X(x)} = \sum_{x \in \Sigma} \mathrm{P}_X(x) = 1$$

Kraft's inequality implies that there is a code with codeword lengths l_x. Then note that
$\sum_{x \in \Sigma} \mathrm{P}_X(x) \lceil -\log \mathrm{P}_X(x) \rceil < H(X) + 1$.

Binary Source Coding Theorem (2/2)

Proof of Lower Bound.

$$\begin{aligned}
\mathrm{E}\left[l(X)\right] &= \sum_x \mathrm{P}_X\left(x\right) l_x \\
&= -\sum_x \mathrm{P}_X\left(x\right) \log 2^{-l_x} \\
&\geq -\sum_x \mathrm{P}_X\left(x\right) \log \mathrm{P}_X\left(x\right) \\
&= H(X)
\end{aligned}$$

Huffman's Code (1/2)

Input: $\{(a_1, p_1), \ldots, (a_n, p_n)\}$.
Output: 0/1-labeled rooted tree.

$\textsc{Huffman}(\{(a_1, p_1), \ldots, (a_n, p_n)\})$

(1) $S \leftarrow \{(a_1, p_1, a_1), \ldots, (a_n, p_n, a_n)\}$
(2) **while** $|S| \geq 2$
(3) Find $(b_i, p_i, t_i), (b_j, p_j, t_j) \in S$ with minimal p_i and p_j.
(4) $S \leftarrow S \setminus \{(b_i, p_i, t_i), (b_j, p_j, t_j)\}$
(5) $S \leftarrow S \cup \{(b_i \| b_j, p_i + p_j, \textsc{Node}(t_i, t_j))\}$
(6) **return** S

Huffman's Code (2/2)

Theorem. Huffman's code is optimal.

Proof idea.
There exists an optimal code where the two least likely symbols are neighbors.

Entropy

Let us turn this expression into a definition.

Definition. Let X be a random variable taking values in $\mathcal{X}$. Then the **entropy** of X is

$$H(X) = -\sum_{x \in \mathcal{X}} \mathrm{P}_X(x) \log \mathrm{P}_X(x) \quad .$$

Examples and intuition are nice, but what we need is a theorem that states that this is **exactly** the right expected length of an optimal code.

Conditional Entropy

Definition. Let (X, Y) be a random variable taking values in $\mathcal{X} \times \mathcal{Y}$. We define **conditional entropy**

$$H(X|y) = -\sum_x \mathrm{P}_{X|Y}\,(x\,|y\,) \log \mathrm{P}_{X|Y}\,(x\,|y\,) \quad \text{and}$$

$$H(X|Y) = \sum_y \mathrm{P}_Y\,(y)\, H(X|y)$$

Note that $H(X|y)$ is simply the ordinary entropy function of a random variable with probability function $\mathrm{P}_{X|Y}\,(\,\cdot\,|y\,)$.

Properties of Entropy

Let X be a random variable taking values in $\mathcal{X}$.

Upper Bound. $H(X) = \mathrm{E}\left[-\log \mathrm{P}_X(X)\right] \leq \log |\mathcal{X}|$.

Chain Rule and Conditioning.

$$
\begin{aligned}
H(X,Y) &= -\sum_{x,y} \mathrm{P}_{X,Y}(x,y) \log \mathrm{P}_{X,Y}(x,y) \\
&= -\sum_{x,y} \mathrm{P}_{X,Y}(x,y) \left(\log \mathrm{P}_Y(y) + \log \mathrm{P}_{X|Y}(x|y) \right) \\
&= -\sum_{y} \mathrm{P}_Y(y) \log \mathrm{P}_Y(y) - \sum_{x,y} \mathrm{P}_{X,Y}(x,y) \log \mathrm{P}_{X|Y}(x|y) \\
&= H(Y) + H(X|Y) \leq H(Y) + H(X)
\end{aligned}
$$

Elementary Number Theory

Greatest Common Divisors

Definition. A common divisor of two integers m and n is an integer d such that $d \mid m$ and $d \mid n$.

Definition. A greatest common divisor (GCD) of two integers m and n is a common divisor d such that every common divisor d' divides d.

Greatest Common Divisors

Definition. A common divisor of two integers m and n is an integer d such that $d \mid m$ and $d \mid n$.

Definition. A greatest common divisor (GCD) of two integers m and n is a common divisor d such that every common divisor d' divides d.

- **The** GCD is the **positive** GCD.

Greatest Common Divisors

Definition. A common divisor of two integers m and n is an integer d such that $d \mid m$ and $d \mid n$.

Definition. A greatest common divisor (GCD) of two integers m and n is a common divisor d such that every common divisor d' divides d.

- **The** GCD is the **positive** GCD.
- We denote the GCD of m and n by $\gcd(m, n)$.

Properties

- $\gcd(m, n) = \gcd(n, m)$
- $\gcd(m, n) = \gcd(m \pm n, n)$
- $\gcd(m, n) = \gcd(m \bmod n, n)$
- $\gcd(m, n) = 2\gcd(m/2, n/2)$ if m and n are even.
- $\gcd(m, n) = \gcd(m/2, n)$ if m is even and n is odd.

Euclidean Algorithm

EUCLIDEAN(m, n)

(1) **while** $n \neq 0$
(2) $\quad t \leftarrow n$
(3) $\quad n \leftarrow m \bmod n$
(4) $\quad m \leftarrow t$
(5) **return** m

Steins Algorithm (Binary GCD Algorithm)

```
STEIN(m, n)
(1)   if m = 0 or n = 0 then return 0
(2)   s ← 0
(3)   while m and n are even
(4)       m ← m/2, n ← n/2, s ← s + 1
(5)   while n is even
(6)       n ← n/2
(7)   while m ≠ 0
(8)       while m is even
(9)           m ← m/2
(10)      if m < n
(11)          SWAP(m, n)
(12)      m ← m − n
(13)      m ← m/2
(14)  return 2^s n
```

Bezout's Lemma

Lemma. There exists integers a and b such that

$$\gcd(m, n) = am + bn \ .$$

Bezout's Lemma

Lemma. There exists integers a and b such that

$$\gcd(m, n) = am + bn \ .$$

Proof. Let $d > \gcd(m, n)$ be the smallest positive integer on the form $d = am + bn$. Write $m = cd + r$ with $0 < r < d$. Then

$$d > r = m - cd = m - c(am + bn) = (1 - ca)m + (-cb)n \ ,$$

a contradiction! Thus, $r = 0$ and $d \mid m$. Similarly, $d \mid n$.

Extended Euclidean Algorithm (Recursive Version)

ExtendedEuclidean(m, n)
(1) **if** $m \bmod n = 0$
(2) **return** $(0, 1)$
(3) **else**
(4) $(x, y) \leftarrow$ ExtendedEuclidean$(n, m \bmod n)$
(5) **return** $(y, x - y\lfloor m/n \rfloor)$

If $(x, y) \leftarrow$ ExtendedEuclidean(m, n) then
$\gcd(m, n) = xm + yn$.

Coprimality (Relative Primality)

Definition. Two integers m and n are coprime if their greatest common divisor is 1.

Fact. If a and n are coprime, then there exists a b such that $ab = 1 \bmod n$.

Coprimality (Relative Primality)

Definition. Two integers m and n are coprime if their greatest common divisor is 1.

Fact. If a and n are coprime, then there exists a b such that $ab = 1 \bmod n$.

Excercise: Why is this so?

Chinese Remainder Theorem (CRT)

Theorem. (Sun Tzu 400 AC) Let $n_1, \ldots, n_k$ be positive pairwise coprime integers and let $a_1, \ldots, a_k$ be integers. Then the equation system

$$\begin{aligned} x &= a_1 \bmod n_1 \\ x &= a_2 \bmod n_2 \\ x &= a_3 \bmod n_3 \\ &\vdots \\ x &= a_k \bmod n_k \end{aligned}$$

has a unique solution in $\{0, \ldots, \prod_i n_i - 1\}$.

Constructive Proof of CRT

1. Set $N = n_1 n_2 \cdot \ldots \cdot n_k$.
2. Find r_i and s_i such that $r_i n_i + s_i \frac{N}{n_i} = 1$ (Bezout).
3. Note that

$$s_i \frac{N}{n_i} = 1 - r_i n_i = \begin{cases} 1 & (\text{mod } n_i) \\ 0 & (\text{mod } n_j) \end{cases} \qquad \text{if } j \neq i$$

4. The solution to the equation system becomes:

$$x = \sum_{i=1}^{k} \left(s_i \frac{N}{n_i} \right) \cdot a_i$$

The Multiplicative Group

The set $\mathbb{Z}_n^* = \{0 \leq a < n \ : \ \gcd(a, n) = 1\}$ forms a group, since:

- **Closure.** It is closed under multiplication modulo n.
- **Associativity.** For $x, y, z \in \mathbb{Z}_n^*$:

$$(xy)z = x(yz) \bmod n \ .$$

- **Identity.** For every $x \in \mathbb{Z}_n^*$:

$$1 \cdot x = x \cdot 1 = x \ .$$

- **Inverse.** For every $a \in \mathbb{Z}_n^*$ exists $b \in \mathbb{Z}_n^*$ such that:

$$ab = 1 \bmod n \ .$$

Lagrange's Theorem

Theorem. If H is a subgroup of a finite group G, then $|H|$ divides $|G|$.

Proof.

1. Define $aH = \{ah : h \in H\}$. This gives an equivalence relation $x \approx y \Leftrightarrow x = yh \wedge h \in H$ on G.
2. The map $\phi_{a,b} : aH \to bH$, defined by $\phi_{a,b}(x) = ba^{-1}x$ is a bijection, so $|aH| = |bH|$ for $a, b \in G$.

Euler's Phi-Function (Totient Function)

Definition. Euler's Phi-function $\phi(n)$ counts the number of integers $0 < a < n$ relatively prime to n.

Euler's Phi-Function (Totient Function)

Definition. Euler's Phi-function $\phi(n)$ counts the number of integers $0 < a < n$ relatively prime to n.

- Clearly: $\phi(p) = p - 1$ when p is prime.

Euler's Phi-Function (Totient Function)

Definition. Euler's Phi-function $\phi(n)$ counts the number of integers $0 < a < n$ relatively prime to n.

- Clearly: $\phi(p) = p - 1$ when p is prime.
- Similarly: $\phi(p^k) = p^k - p^{k-1}$ when p is prime and $k > 1$.

Euler's Phi-Function (Totient Function)

Definition. Euler's Phi-function $\phi(n)$ counts the number of integers $0 < a < n$ relatively prime to n.

- Clearly: $\phi(p) = p - 1$ when p is prime.
- Similarly: $\phi(p^k) = p^k - p^{k-1}$ when p is prime and $k > 1$.
- In general: $\phi\left(\prod_i p_i^{k_i}\right) = \prod_i \left(p_i^k - p_i^{k-1}\right)$.

Euler's Phi-Function (Totient Function)

Definition. Euler's Phi-function $\phi(n)$ counts the number of integers $0 < a < n$ relatively prime to n.

- Clearly: $\phi(p) = p - 1$ when p is prime.
- Similarly: $\phi(p^k) = p^k - p^{k-1}$ when p is prime and $k > 1$.
- In general: $\phi\left(\prod_i p_i^{k_i}\right) = \prod_i \left(p_i^k - p_i^{k-1}\right)$.

Excercise: How does this follow from CRT?

Fermat's and Euler's Theorems

Theorem. (Fermat) If $b \in \mathbb{Z}_p^*$ and p is prime, then $b^{p-1} = 1 \bmod p$.

Theorem. (Euler) If $b \in \mathbb{Z}_n^*$, then $b^{\phi(n)} = 1 \bmod n$.

Fermat's and Euler's Theorems

Theorem. (Fermat) If $b \in \mathbb{Z}_p^*$ and p is prime, then $b^{p-1} = 1 \bmod p$.

Theorem. (Euler) If $b \in \mathbb{Z}_n^*$, then $b^{\phi(n)} = 1 \bmod n$.

Proof. Note that $|\mathbb{Z}_n^*| = \phi(n)$. b generates a subgroup $\langle b \rangle$ of $\mathbb{Z}_n^*$, so $|\langle b \rangle|$ divides $\phi(n)$ and $b^{\phi(n)} = 1 \bmod n$.

Multiplicative Group of a Prime Order Field

Definition. A group G is called **cyclic** if there exists an element g such that each element in G is on the form g^x for some integer x.

Theorem. If p is prime, then $\mathbb{Z}_p^*$ is cyclic.

Multiplicative Group of a Prime Order Field

Definition. A group G is called **cyclic** if there exists an element g such that each element in G is on the form g^x for some integer x.

Theorem. If p is prime, then $\mathbb{Z}_p^*$ is cyclic.

Every group of prime order is cyclic. Why?

Multiplicative Group of a Prime Order Field

Definition. A group G is called **cyclic** if there exists an element g such that each element in G is on the form g^x for some integer x.

Theorem. If p is prime, then $\mathbb{Z}_p^*$ is cyclic.

Every group of prime order is cyclic. Why?

Why is there no cyclic multiplicative group $\mathbb{Z}_p^*$, with prime p, except the trivial case $\mathbb{Z}_2^*$?

Multiplicative Group of a Prime Order Field

Definition. A group G is called **cyclic** if there exists an element g such that each element in G is on the form g^x for some integer x.

Theorem. If p is prime, then $\mathbb{Z}_p^*$ is cyclic.

Every group of prime order is cyclic. Why?

Why is there no cyclic multiplicative group $\mathbb{Z}_p^*$, with prime p, except the trivial case $\mathbb{Z}_2^*$?

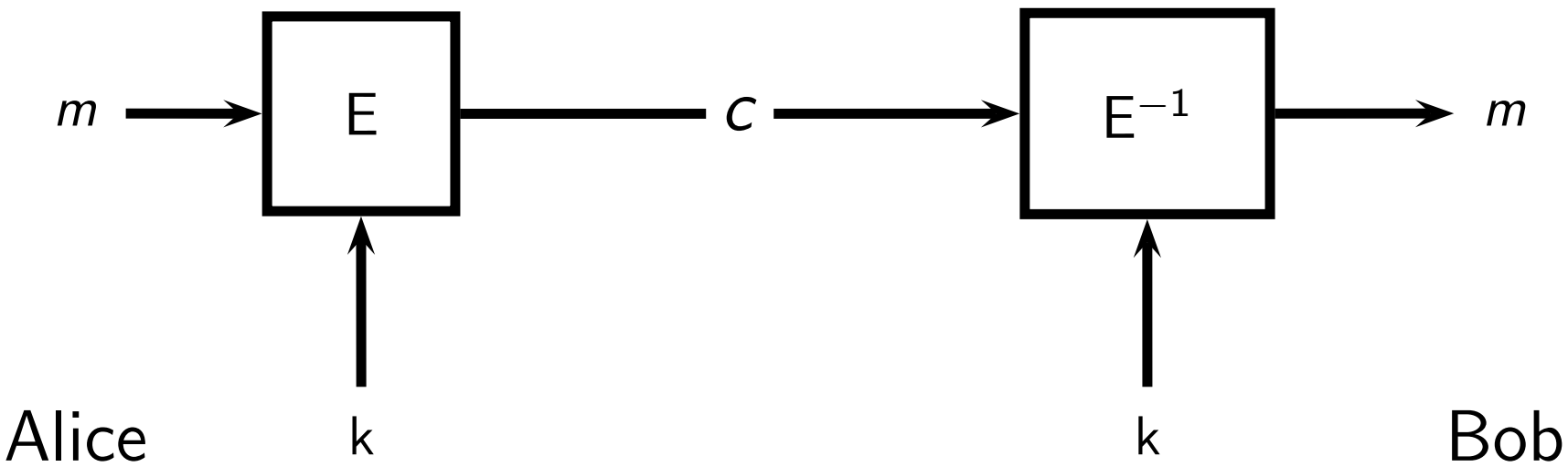
Keep in mind the difference between:

- $\mathbb{Z}_p$ with *prime order* as an *additive group*,
- $\mathbb{Z}_p^*$ with *non-prime order* as a *multiplicative group*.
- group G_p of *prime order*.

Public-Key Cryptography

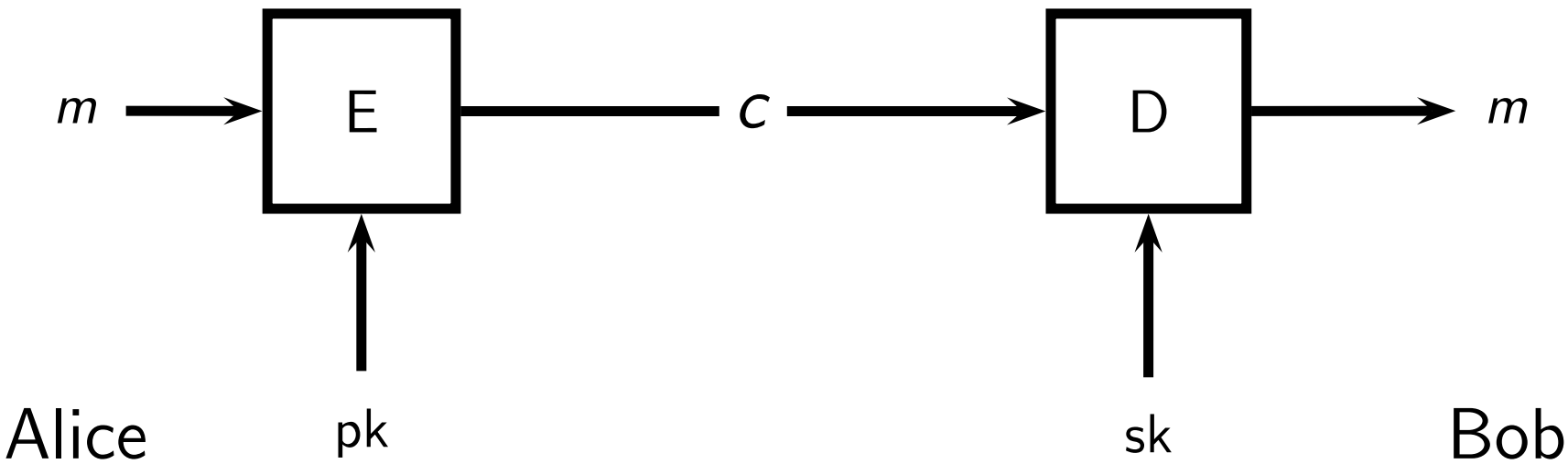
Cipher (Symmetric Cryptosystem)

$c = \mathsf{E}_k(m)$ $\qquad$ $m = \mathsf{E}_k^{-1}(c)$



Public-Key Cryptosystem

$c = \mathsf{E}_{\mathsf{pk}}(m)$ $\qquad$ $m = \mathsf{D}_{\mathsf{sk}}(c)$



History of Public-Key Cryptography

Public-key cryptography was discovered:

- By Ellis, Cocks, and Williamson at the Government Communications Headquarters (GCHQ) in the UK in the early 1970s (not public until 1997).
- Independently by Merkle in 1974 (Merkle's puzzles).
- Independently in its discrete-logarithm based form by Diffie and Hellman in 1977, and instantiated in 1978 (key-exchange).
- Independently in its factoring-based form by Rivest, Shamir and Adleman in 1977.

Public-Key Cryptography

Definition. A public-key cryptosystem is a tuple $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ where,

- Gen is a **probabilistic key generation algorithm** that outputs key pairs $(\mathsf{pk}, \mathsf{sk})$,
- E is a (possibly probabilistic) **encryption algorithm** that given a public key pk and a message m in the plaintext space $\mathcal{M}_{\mathsf{pk}}$ outputs a ciphertext c, and
- D is a **decryption algorithm** that given a secret key sk and a ciphertext c outputs a plaintext m,

such that $\mathsf{D}_{\mathsf{sk}}(\mathsf{E}_{\mathsf{pk}}(m)) = m$ for every $(\mathsf{pk}, \mathsf{sk})$ and $m \in \mathcal{M}_{\mathsf{pk}}$.

RSA

The RSA Cryptosystem (1/2)

Key Generation.

- Choose $n/2$-bit primes p and q randomly and define $N = pq$.
- Choose e in $\mathbb{Z}^*_{\phi(N)}$ and compute $d = e^{-1} \bmod \phi(N)$.
- Output the key pair $((N, e), (p, q, d))$, where (N, e) is the public key and (p, q, d) is the secret key.

The RSA Cryptosystem (2/2)

Encryption. Encrypt a plaintext $m \in \mathbb{Z}_N^*$ by computing

$$c = m^e \bmod N \ .$$

Decryption. Decrypt a ciphertext c by computing

$$m = c^d \bmod N \ .$$

Why Does It Work?

$$(m^e \bmod N)^d \bmod N = m^{ed} \bmod N$$

Why Does It Work?

$$
\begin{aligned}
(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\
&= m^{1+t\phi(N)} \bmod N
\end{aligned}
$$

Why Does It Work?

$$
\begin{aligned}
(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\
&= m^{1+t\phi(N)} \bmod N \\
&= m^1 \cdot \left(m^{\phi(N)}\right)^t \bmod N
\end{aligned}
$$

Why Does It Work?

$$
\begin{aligned}
(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\
&= m^{1+t\phi(N)} \bmod N \\
&= m^1 \cdot \left(m^{\phi(N)}\right)^t \bmod N \\
&= m \cdot 1^t \bmod N
\end{aligned}
$$

Why Does It Work?

$$
\begin{aligned}
(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\
&= m^{1+t\phi(N)} \bmod N \\
&= m^1 \cdot \left(m^{\phi(N)}\right)^t \bmod N \\
&= m \cdot 1^t \bmod N \\
&= m \bmod N
\end{aligned}
$$

Implementing RSA

- Modular arithmetic.
- Greatest common divisor.
- Primality test.

Modular Arithmetic (1/3)

Basic operations on $O(n)$-bit integers using "school book" implementations.

Operation	Running time
Addition	$O(n)$
Subtraction	$O(n)$
Multiplication	$O(n^2)$
Modular reduction	$O(n^2)$
Greatest common divisor	$O(n^2)$

Modular Arithmetic (1/3)

Basic operations on $O(n)$-bit integers using "school book" implementations.

Operation	Running time
Addition	$O(n)$
Subtraction	$O(n)$
Multiplication	$O(n^2)$
Modular reduction	$O(n^2)$
Greatest common divisor	$O(n^2)$

Optimal algorithms for multiplication and modular reduction are much faster.

Modular Arithmetic (1/3)

Basic operations on $O(n)$-bit integers using "school book" implementations.

Operation	Running time
Addition	$O(n)$
Subtraction	$O(n)$
Multiplication	$O(n^2)$
Modular reduction	$O(n^2)$
Greatest common divisor	$O(n^2)$

Optimal algorithms for multiplication and modular reduction are much faster.

What about modular exponentiation?

Modular Arithmetic (2/3)

Square-and-Multiply.

SquareAndMultiply(x, e, N)

1 $z \leftarrow 1$
2 $i =$index of most significant one
3 **while** $i \geq 0$
do
4 $z \leftarrow z \cdot z \bmod N$
5 **if** $e_i = 1$
then $z \leftarrow z \cdot x \bmod N$
6 $i \leftarrow i - 1$
7 **return** z

Modular Arithmetic (3/3)

Although the basic is the same, the most efficient algorithms for exponentiation is faster.

Computing $g^{x_1}, \ldots, g^{x_k}$ can be done much faster!

Computing $\prod_{i \in [k]} g_i^{x_i}$ can be done much faster!

Modular Arithmetic (3/3)

Although the basic is the same, the most efficient algorithms for exponentiation is faster.

Computing $g^{x_1}, \ldots, g^{x_k}$ can be done much faster!

Computing $\prod_{i \in [k]} g_i^{x_i}$ can be done much faster!

How about side channel attacks?

Prime Number Theorem

The primes are relatively dense.

Prime Number Theorem

The primes are relatively dense.

Theorem. Let $\pi(m)$ denote the number of primes $0 < p \leq m$. Then

$$\lim_{m \to \infty} \frac{\pi(m)}{\frac{m}{\ln m}} = 1 \ .$$

Prime Number Theorem

The primes are relatively dense.

Theorem. Let $\pi(m)$ denote the number of primes $0 < p \leq m$. Then

$$\lim_{m \to \infty} \frac{\pi(m)}{\frac{m}{\ln m}} = 1 \ .$$

To generate a random prime, we repeatedly pick a random integer m and check if it is prime. It should be prime with probability close to $1/\ln m$ in a sufficently large interval.

Legendre Symbol (1/2)

Definition. Given an odd integer $b \geq 3$, an integer a is called a **quadratic residue** modulo b if there exists an integer x such that $a = x^2 \bmod b$.

Definition. The **Legendre Symbol** of an integer a modulo an **odd prime** p is defined by

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a = 0 \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}.$$

Legendre Symbol (2/2)

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \ .$$

Legendre Symbol (2/2)

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \ .$$

Proof.

- If $a = y^2 \bmod p$, then $a^{(p-1)/2} = y^{p-1} = 1 \bmod p$.

Legendre Symbol (2/2)

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \ .$$

Proof.

- If $a = y^2 \bmod p$, then $a^{(p-1)/2} = y^{p-1} = 1 \bmod p$.
- If $a^{(p-1)/2} = 1 \bmod p$ and b generates $\mathbb{Z}_p^*$, then $a^{(p-1)/2} = b^{x(p-1)/2} = 1 \bmod p$ for some x. Since b is a generator, $(p-1) \mid x(p-1)/2$ and x must be even.

Legendre Symbol (2/2)

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \ .$$

Proof.

- If $a = y^2 \bmod p$, then $a^{(p-1)/2} = y^{p-1} = 1 \bmod p$.
- If $a^{(p-1)/2} = 1 \bmod p$ and b generates $\mathbb{Z}_p^*$, then $a^{(p-1)/2} = b^{x(p-1)/2} = 1 \bmod p$ for some x. Since b is a generator, $(p-1) \mid x(p-1)/2$ and x must be even.
- If a is a non-residue, then $a^{(p-1)/2} \neq 1 \bmod p$, but $\left(a^{(p-1)/2}\right)^2 = 1 \bmod p$, so $a^{(p-1)/2} = -1 \bmod p$.

Jacobi Symbol

Definition. The **Jacobi Symbol** of an integer a modulo an odd integer $b = \prod_i p_i^{e_i}$, with p_i prime, is defined by

$$\left(\frac{a}{b}\right) = \prod_i \left(\frac{a}{p_i}\right)^{e_i} .$$

Note that we can have $\left(\frac{a}{b}\right) = 1$ even when a is a non-residue modulo b.

Properties of the Jacobi Symbol

Basic Properties.

$$\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$$
$$\left(\frac{ac}{b}\right) = \left(\frac{a}{b}\right)\left(\frac{c}{b}\right) \quad .$$

Law of Quadratic Reciprocity. If a and b are odd integers, then

$$\left(\frac{a}{b}\right) = (-1)^{\frac{(a-1)(b-1)}{4}} \left(\frac{b}{a}\right) \quad .$$

Supplementary Laws. If b is an odd integer, then

$$\left(\frac{-1}{b}\right) = (-1)^{\frac{b-1}{2}} \quad \text{and} \quad \left(\frac{2}{b}\right) = (-1)^{\frac{b^2-1}{8}} \quad .$$

Computing the Jacobi Symbol (1/2)

The following assumes that $a \geq 0$ and that $b \geq 3$ is odd.

JACOBI(a, b)

(1) **if** $a < 2$
(2) **return** a
(3) $s \leftarrow 1$
(4) **while** a is even
(5) $s \leftarrow s \cdot (-1)^{\frac{1}{8}(b^2-1)}$
(6) $a \leftarrow a/2$
(7) **if** $a < b$
(8) SWAP(a,b)
(9) $s \leftarrow s \cdot (-1)^{\frac{1}{4}(a-1)(b-1)}$
(10) **return** $s \cdot$ JACOBI($a \bmod b, b$)

Solovay-Strassen Primality Test (1/2)

The following assumes that $n \geq 3$.

SolovayStrassen(n, r)

(1) **for** $i = 1$ **to** r
(2) Choose $0 < a < n$ randomly.
(3) **if** $\left(\frac{a}{n}\right) = 0$ or $\left(\frac{a}{n}\right) \neq a^{(n-1)/2} \bmod n$
(4) **return** *composite*
(5) **return** *probably prime*

Solovay-Strassen Primality Test (2/2)

Analysis.

- If n is prime, then $0 \neq \left(\frac{a}{n}\right) = a^{(n-1)/2} \bmod n$ for all $0 < a < n$, so we never claim that a prime is composite.

Solovay-Strassen Primality Test (2/2)

Analysis.

- If n is prime, then $0 \neq \left(\frac{a}{n}\right) = a^{(n-1)/2} \bmod n$ for all $0 < a < n$, so we never claim that a prime is composite.
- If $\left(\frac{a}{n}\right) = 0$, then $\left(\frac{a}{p}\right) = 0$ for some prime factor p of n. Thus, $p \mid a$ and n is composite, so we never wrongly return from within the loop.

Solovay-Strassen Primality Test (2/2)

Analysis.

- If n is prime, then $0 \neq \left(\frac{a}{n}\right) = a^{(n-1)/2} \bmod n$ for all $0 < a < n$, so we never claim that a prime is composite.
- If $\left(\frac{a}{n}\right) = 0$, then $\left(\frac{a}{p}\right) = 0$ for some prime factor p of n. Thus, $p \mid a$ and n is composite, so we never wrongly return from within the loop.
- At most half of all elements a in $\mathbb{Z}_n^*$ have the property that

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

More On Primality Tests

- The Miller-Rabin test is faster.
- Testing many primes can be done faster than testing each separately
- Those are *probabilistic* primality tests, but there is a *deterministic* test, so Primes are in P!

Security of RSA

Factoring

The obvious way to break RSA is to factor the public modulus N and recover the prime factors p and q.

- The number field sieve factors N in time

$$O\left(e^{(1.92+o(1))\left((\ln N)^{1/3}+(\ln\ln N)^{2/3}\right)}\right) .$$

- The elliptic curve method factors N in time

$$O\left(e^{(1+o(1))\sqrt{2\ln p\ln\ln p}}\right) .$$

Factoring

The obvious way to break RSA is to factor the public modulus N and recover the prime factors p and q.

- The number field sieve factors N in time

$$O\left(e^{(1.92+o(1))\left((\ln N)^{1/3}+(\ln\ln N)^{2/3}\right)}\right) .$$

- The elliptic curve method factors N in time

$$O\left(e^{(1+o(1))\sqrt{2\ln p\ln\ln p}}\right) .$$

Note that the latter only depends on the size of p!

Small Encryption Exponents

Suppose that $e = 3$ is used by all parties as encryption exponent.

- **Small Message.** If m is small, then $m^e < N$. Thus, **no reduction takes place**, and m can be computed in $\mathbb{Z}$ by taking the eth root.

Small Encryption Exponents

Suppose that $e = 3$ is used by all parties as encryption exponent.

- **Small Message.** If m is small, then $m^e < N$. Thus, **no reduction takes place**, and m can be computed in $\mathbb{Z}$ by taking the eth root.
- **Identical Plaintexts.** If a message m is encrypted under moduli N_1, N_2, N_3, and N_4 as c_1, c_2, c_3, and c_3, then CRT implies a $c \in \mathbb{Z}^*_{N_1 N_2 N_3 N_4}$ such that $c = c_i \bmod N_i$ and $c = m^e \bmod N_1 N_2 N_3 N_4$ with $m < N_i$.

Additional Caveats

- **Identical Moduli.** If a message m is encrypted as c_1 and c_2 using distinct encryption exponents e_1 and e_2 with $\gcd(e_1, e_2) = 1$, and a modulus N, then we can find a, b such that $ae_1 + be_2 = 1$ and $m = c_1^a c_2^b \bmod N$.

Additional Caveats

- **Identical Moduli.** If a message m is encrypted as c_1 and c_2 using distinct encryption exponents e_1 and e_2 with $\gcd(e_1, e_2) = 1$, and a modulus N, then we can find a, b such that $ae_1 + be_2 = 1$ and $m = c_1^a c_2^b \bmod N$.
- **Reiter-Franklin Attack.** If e is small then encryptions of m and $f(m)$ for a polynomial $f \in \mathbb{Z}_N[x]$ allows efficient computation of m.

Additional Caveats

- **Identical Moduli.** If a message m is encrypted as c_1 and c_2 using distinct encryption exponents e_1 and e_2 with $\gcd(e_1, e_2) = 1$, and a modulus N, then we can find a, b such that $ae_1 + be_2 = 1$ and $m = c_1^a c_2^b \bmod N$.
- **Reiter-Franklin Attack.** If e is small then encryptions of m and $f(m)$ for a polynomial $f \in \mathbb{Z}_N[x]$ allows efficient computation of m.
- **Wiener's Attack.** If $3d < N^{1/4}$ and $q < p < 2q$, then N can be factored in polynomial time with good probability.

Factoring From Order of Multiplicative Group

Given N and $\phi(N)$, we can find p and q by solving

$$N = pq$$
$$\phi(N) = (p-1)(q-1)$$

Factoring From Encryption & Decryption Exponents (1/3)

- If $N = pq$ with p and q prime, then the CRT implies that

$$x^2 = 1 \bmod N$$

has **four distinct solutions** in $\mathbb{Z}_N^*$, and **two** of these are **non-trivial**, i.e., distinct from ± 1.

Factoring From Encryption & Decryption Exponents (1/3)

- If $N = pq$ with p and q prime, then the CRT implies that

$$x^2 = 1 \bmod N$$

has **four distinct solutions** in $\mathbb{Z}_N^*$, and **two** of these are **non-trivial**, i.e., distinct from ± 1.

- If x is a non-trivial root, then

$$(x-1)(x+1) = tN$$

but $N \nmid (x-1), (x+1)$, so

$$\gcd(x-1, N) > 1 \quad \text{and} \quad \gcd(x+1, N) > 1 \ .$$

Factoring From Encryption & Decryption Exponents (2/3)

- The encryption & decryption exponents satisfy

$$ed = 1 \bmod \phi(N) \ ,$$

so if we have $ed - 1 = 2^s r$ with r odd, then

$$\begin{aligned} (p-1) &= 2^{s_p} r_p \ \text{ which divides } \ 2^s r \quad \text{and} \\ (q-1) &= 2^{s_q} r_q \ \text{ which divides } \ 2^s r \ . \end{aligned}$$

- If $v \in \mathbb{Z}_N^*$ is random, then $w = v^r$ is random in the subgroup of elements with order 2^i for some $0 \leq i \leq \max\{s_p, s_q\}$.

Factoring From Encryption & Decryption Exponents (3/3)

Suppose $s_p \geq s_q$. Then for some $0 < i < s_p$,

$$w^{2^i} = \pm 1 \bmod q$$

and

$$w^{2^i} \bmod p$$

is uniformly distributed in $\{1, -1\}$.

Conclusion.
$w^{2^i} \pmod N$ is a non-trivial root of 1 with probability $1/2$, which allows us to factor N.

Semantic Security

Semantic Security (1/3)

- RSA clearly provides some kind of "security", but it is clear that we need to be more careful with what we ask for.

Semantic Security (1/3)

- RSA clearly provides some kind of "security", but it is clear that we need to be more careful with what we ask for.
- Intuitively, we want to leak no information of the encrypted plaintext.

Semantic Security (1/3)

- RSA clearly provides some kind of "security", but it is clear that we need to be more careful with what we ask for.
- Intuitively, we want to leak no **knowledge** of the encrypted plaintext.

Semantic Security (1/3)

- RSA clearly provides some kind of "security", but it is clear that we need to be more careful with what we ask for.
- Intuitively, we want to leak no **knowledge** of the encrypted plaintext.
- In other words, no function of the plaintext can efficiently be guessed notably better from its ciphertext than without it.

  Idea! Define only **lack** of knowledge and not what knowledge actually is.

Semantic Security (2/3)

$\mathbf{Exp}^{b}_{\mathcal{CS},A}$ **(Semantic Security Experiment).**

1. **Generate Public Key.** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$.
2. **Adversarial Choice of Messages.** $(m_0, m_1, s) \leftarrow A(\mathsf{pk})$.
3. **Guess Message.** Return the first bit of $A(\mathsf{E}_{\mathsf{pk}}(m_b), s)$.

Semantic Security (2/3)

$\mathrm{Exp}^{b}_{\mathcal{CS},A}$ **(Semantic Security Experiment).**

1. **Generate Public Key.** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$.
2. **Adversarial Choice of Messages.** $(m_0, m_1, s) \leftarrow A(\mathsf{pk})$.
3. **Guess Message.** Return the first bit of $A(\mathsf{E}_{\mathsf{pk}}(m_b), s)$.

Definition. A cryptosystem $\mathcal{CS} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is said to be **semantically secure** if for every polynomial time algorithm A

$$|\Pr[\mathrm{Exp}^{0}_{\mathcal{CS},A} = 1] - \Pr[\mathrm{Exp}^{1}_{\mathcal{CS},A} = 1]|$$

is negligible.

Semantic Security (3/3)

Every semantically secure cryptosystem must be probabilistic!

Semantic Security (3/3)

Every semantically secure cryptosystem must be probabilistic!

Theorem. Suppose that $\mathcal{CS} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is a semantically secure cryptosystem.

Then the related cryptosystem where a $t(n)$-list of messages, with $t(n)$ polynomial, is encrypted by **repeated independent encryption** of each component using the **same public key** is also semantically secure.

Semantic Security (3/3)

Every semantically secure cryptosystem must be probabilistic!

Theorem. Suppose that $\mathcal{CS} = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is a semantically secure cryptosystem.

Then the related cryptosystem where a $t(n)$-list of messages, with $t(n)$ polynomial, is encrypted by **repeated independent encryption** of each component using the **same public key** is also semantically secure.

Semantic security is useful!

ROM-RSA

The RSA Assumption

Definition. The RSA assumption states that if:

1. $N = pq$ factors into two randomly chosen primes p and q of the same bit-size,
2. e is in $\mathbb{Z}^*_{\phi(N)}$,
3. m is randomly chosen in $\mathbb{Z}^*_N$,

then for every polynomial time algorithm A

$$\Pr[A(N, e, m^e \bmod N) = m]$$

is negligible.

Semantically Secure ROM-RSA (1/2)

Suppose that $f : \{0,1\}^n \to \{0,1\}^n$ is a randomly chosen function (a random oracle).

- **Key Generation.** Choose a random RSA key pair $((N, e), (p, q, d))$, with $\log_2 N = n$.
- **Encryption.** Encrypt a plaintext $m \in \{0,1\}^n$ by choosing $r \in \mathbb{Z}_N^*$ randomly and computing

$$(u, v) = (r^e \bmod N, f(r) \oplus m) \ .$$

- **Decryption.** Decrypt a ciphertext (u, v) by

$$m = v \oplus f(u^d \bmod N) \ .$$

Semantically Secure RSA in the ROM (2/2)

- We increase the ciphertext size by a factor of two.
- Our analysis is in the random oracle model, **which is unsound!**

Semantically Secure RSA in the ROM (2/2)

- We increase the ciphertext size by a factor of two.
- Our analysis is in the random oracle model, **which is unsound!**

Solutions.

- Using a "optimal" padding the first problem can be reduced. See standard OAEP+.

Semantically Secure RSA in the ROM (2/2)

- We increase the ciphertext size by a factor of two.
- Our analysis is in the random oracle model, **which is unsound!**

Solutions.

- Using a "optimal" padding the first problem can be reduced. See standard OAEP+.
- Using a scheme with much lower rate, the second problem can be removed.

Rabin

Rabin's Cryptosystem (1/3)

Key Generation.

- Choose n-bit primes p and q such that $p, q = 3 \bmod 4$ randomly and define $N = pq$.
- Output the key pair $(N, (p, q))$, where N is the public key and (p, q) is the secret key.

Rabin's Cryptosystem (2/3)

Encryption. Encrypt a plaintext m by computing

$$c = m^2 \bmod N \ .$$

Decryption. Decrypt a ciphertext c by computing

$$m = \sqrt{c} \bmod N \ .$$

Rabin's Cryptosystem (2/3)

Encryption. Encrypt a plaintext m by computing

$$c = m^2 \bmod N \ .$$

Decryption. Decrypt a ciphertext c by computing

$$m = \sqrt{c} \bmod N \ .$$

There are **four** roots, so which one should be used?

Rabin's Cryptosystem (3/3)

Suppose y is a quadratic residue modulo p.

$$\begin{aligned}\left(\pm y^{(p+1)/4}\right)^2 &= y^{(p+1)/2} \bmod p \\ &= y^{(p-1)/2} y \bmod p \\ &= \left(\frac{y}{p}\right) y \\ &= y \bmod p\end{aligned}$$

In Rabin's cryptosystem:

- Find roots for $y_p = y \bmod p$ and $y_q = y \bmod q$.
- Combine roots to get the four roots modulo N. Choose the "right" root and output the plaintext.

Security of Rabin's Cryptosystem

Theorem. Breaking Rabin's cryptosystem is equivalent to factoring.

Idea.

1. Choose random element r.
2. Hand r^2 mod N to adversary.
3. Consider outputs r' from the adversary such that $(r')^2 = r^2 \bmod N$. Then $r' \neq \pm r \bmod N$, with probability $1/2$, in which case $\gcd(r' - r, N)$ gives a factor of N.

A Goldwasser-Micali Variant of Rabin

Theorem [CG98]. If factoring is hard and r is a random quadratic residue modulo N, then for every polynomial time algorithm A

$$\Pr[A(N, r^2 \bmod N) = \mathrm{lsb}(r)]$$

is negligible.

- **Encryption.** Encrypt a plaintext $m \in \{0,1\}$ by choosing a random quadratic residue r modulo N and computing

$$(u, v) = (r^2 \bmod N, \mathrm{lsb}(r) \oplus m) \ .$$

- **Decryption.** Decrypt a ciphertext (u, v) by

$$m = v \oplus \mathrm{lsb}(\sqrt{u}) \quad \text{where } \sqrt{u} \text{ is a quadratic residue} \ .$$

Diffie-Hellman

Diffie-Hellman Key Exchange (1/3)

Diffie and Hellman asked themselves:

How can two parties efficiently agree on a secret key using only **public** communication?

Diffie-Hellman Key Exchange (2/3)

Construction.
Let G be a cyclic group of order q with generator g.

1.
   - Alice picks $a \in \mathbb{Z}_q$ randomly, computes $y_a = g^a$ and hands y_a to Bob.
   - Bob picks $b \in \mathbb{Z}_q$ randomly, computes $y_b = g^b$ and hands y_b to Alice.
2.
   - Alice computes $k = y_b^a$.
   - Bob computes $k = y_a^b$.
3. The joint secret key is k.

Diffie-Hellman Key Exchange (3/3)

Problems.

- Susceptible to man-in-the-middle attack without authentication.
- How do we map a random element $k \in G$ to a random symmetric key in $\{0,1\}^n$?

The El Gamal Cryptosystem (1/2)

Definition. Let G be a cyclic group of order q with generator g.

- The **key generation** algorithm chooses a random element $x \in \mathbb{Z}_q$ as the private key and defines the public key as

$$y = g^x \ .$$

- The **encryption** algorithm takes a message $m \in G$ and the public key y, chooses $r \in \mathbb{Z}_q$, and outputs the pair

$$(u, v) = \mathsf{E}_y(m, r) = (g^r, y^r m) \ .$$

- The **decryption** algorithm takes a ciphertext (u, v) and the secret key and outputs

$$m = \mathsf{D}_x(u, v) = vu^{-x} \ .$$

The El Gamal Cryptosystem (2/2)

- El Gamal is essentially Diffie-Hellman + OTP.
- Homomorphic property (with public key y)

$$\mathrm{E}_y(m_0, r_0)\mathrm{E}_y(m_1, r_1) = \mathrm{E}_y(m_0 m_1, r_0 + r_1) \ .$$

This property is very important in the construction of cryptographic protocols!

Discrete Logarithm (1/2)

Definition. Let G be a cyclic group of order q and let g be a generator G. The **discrete logarithm** of $y \in G$ in the basis g (written $\log_g y$) is defined as the unique $x \in \{0, 1, \ldots, q-1\}$ such that

$$y = g^x \quad .$$

Compare with a "normal" logarithm! ($\ln y = x$ iff $y = e^x$)

Discrete Logarithm (2/2)

Example. 7 is a generator of $\mathbb{Z}_{12}$ additively, since $\gcd(7, 12) = 1$.

What is $\log_7 3$?

Discrete Logarithm (2/2)

Example. 7 is a generator of $\mathbb{Z}_{12}$ additively, since $\gcd(7, 12) = 1$.

What is $\log_7 3$? $(9 \cdot 7 = 63 = 3 \bmod 12$, so $\log_7 3 = 9)$

Discrete Logarithm (2/2)

Example. 7 is a generator of $\mathbb{Z}_{12}$ additively, since $\gcd(7, 12) = 1$.

What is $\log_7 3$? $(9 \cdot 7 = 63 = 3 \bmod 12$, so $\log_7 3 = 9)$

Example. 7 is a generator of $\mathbb{Z}_{13}^*$.

What is $\log_7 9$?

Discrete Logarithm (2/2)

Example. 7 is a generator of $\mathbb{Z}_{12}$ additively, since $\gcd(7, 12) = 1$.

What is $\log_7 3$? ($9 \cdot 7 = 63 = 3 \bmod 12$, so $\log_7 3 = 9$)

Example. 7 is a generator of $\mathbb{Z}_{13}^*$.

What is $\log_7 9$? ($7^4 = 9 \bmod 13$, so $\log_7 9 = 4$)

Discrete Logarithm Assumption

Let G_{q_n} be a cyclic group of prime order q_n such that $\lfloor \log_2 q_n \rfloor = n$ for $n = 2, 3, 4, \ldots$, and denote the family $\{G_{q_n}\}_{n \in \mathbb{N}}$ by G.

Definition. The **Discrete Logarithm (DL) Assumption** in G states that if generators g_n and y_n of G_{q_n} are randomly chosen, then for every polynomial time algorithm A

$$\Pr\left[A(g_n, y_n) = \log_{g_n} y_n\right]$$

is negligible.

Discrete Logarithm Assumption

Let G_{q_n} be a cyclic group of prime order q_n such that $\lfloor \log_2 q_n \rfloor = n$ for $n = 2, 3, 4, \ldots$, and denote the family $\{G_{q_n}\}_{n \in \mathbb{N}}$ by G.

Definition. The **Discrete Logarithm (DL) Assumption** in G states that if generators g and y of G are randomly chosen, then for every polynomial time algorithm A

$$\Pr\left[A(g, y) = \log_g y\right]$$

is negligible.

We usually remove the indices from our notation!

Diffie-Hellman Assumption

Definition. Let g be a generator of G. The **Diffie-Hellman (DH) Assumption** in G states that if $a, b \in \mathbb{Z}_q$ are randomly chosen, then for every polynomial time algorithm A

$$\Pr\left[A(g^a, g^b) = g^{ab}\right]$$

is negligible.

Decision Diffie-Hellman Assumption

Definition. Let g be a generator of G. The **Decision Diffie-Hellman (DDH) Assumption** in G states that if $a, b, c \in \mathbb{Z}_q$ are randomly chosen, then for every polynomial time algorithm A

$$\left|\Pr\left[A(g^a, g^b, g^{ab}) = 1\right] - \Pr\left[A(g^a, g^b, g^c) = 1\right]\right|$$

is negligible.

Relating DL Assumptions

- Computing discrete logarithms is at least as hard as computing a Diffie-Hellman element g^{ab} from g^a and g^b.
- Computing a Diffie-Hellman element g^{ab} from g^a and g^b is at least as hard as distinguishing a Diffie-Hellman triple (g^a, g^b, g^{ab}) from a random triple (g^a, g^b, g^c).
- In most groups where the DL assumption is conjectured, DH and DDH assumptions are conjectured as well.
- There exists special elliptic curves where DDH problem is easy, but DH assumption is conjectured!

Security of El Gamal

- Finding the secret key is equivalent to DL problem.
- Finding the plaintext from the ciphertext and the public key and is equivalent to DH problem.
- The semantic security of El Gamal is equivalent to DDH problem.

Brute Force and Shank's

Let G be a cyclic group of order q and g a generator. We wish to compute $\log_g y$.

- **Brute Force.** $O(q)$
- **Shanks.** Time and **Space** $O(\sqrt{q})$.
  1. Set $z = g^m$ (think of m as $m = \sqrt{q}$).
  2. Compute z^i for $0 \leq i \leq q/m$.
  3. Find $0 \leq j \leq m$ and $0 \leq i \leq q/m$ such that $yg^j = z^i$ and output $x = mi - j$.

Birthday Paradox

Lemma. Let $q_0, \ldots, q_k$ be randomly chosen in a set S. Then

1. the probability that $q_i = q_j$ for some $i \neq j$ is approximately $1 - e^{-\frac{k^2}{2s}}$, where $s = |S|$, and
2. with $k \approx \sqrt{-2s \ln(1-\delta)}$ we have a collision-probability of δ.

Proof.

$$\left(\frac{s-1}{s}\right)\left(\frac{s-2}{s}\right)\cdot\ldots\cdot\left(\frac{s-k}{s}\right) \approx \prod_{i=1}^{k} e^{-\frac{i}{s}} \approx e^{-\frac{k^2}{2s}} \quad .$$

Pollard-ρ (1/2)

Partition G into S_1, S_2, and S_3 "randomly".

- Generate "random" sequence $\alpha_0, \alpha_1, \alpha_2 \ldots$

$$\alpha_0 = g$$

$$\alpha_i = \begin{cases} \alpha_{i-1} g & \text{if } \alpha_{i-1} \in S_1 \\ \alpha_{i-1}^2 & \text{if } \alpha_{i-1} \in S_2 \\ \alpha_{i-1} y & \text{if } \alpha_{i-1} \in S_3 \end{cases}$$

Pollard-ρ (1/2)

Partition G into S_1, S_2, and S_3 "randomly".

- Generate "random" sequence $\alpha_0, \alpha_1, \alpha_2 \ldots$

$$\alpha_0 = g$$

$$\alpha_i = \begin{cases} \alpha_{i-1} g & \text{if } \alpha_{i-1} \in S_1 \\ \alpha_{i-1}^2 & \text{if } \alpha_{i-1} \in S_2 \\ \alpha_{i-1} y & \text{if } \alpha_{i-1} \in S_3 \end{cases}$$

- Each $\alpha_i = g^{a_i} y^{b_i}$, where $a_i, b_i \in \mathbb{Z}_q$ are known!

Pollard-ρ (1/2)

Partition G into S_1, S_2, and S_3 "randomly".

- Generate "random" sequence $\alpha_0, \alpha_1, \alpha_2 \ldots$

$$\alpha_0 = g$$

$$\alpha_i = \begin{cases} \alpha_{i-1} g & \text{if } \alpha_{i-1} \in S_1 \\ \alpha_{i-1}^2 & \text{if } \alpha_{i-1} \in S_2 \\ \alpha_{i-1} y & \text{if } \alpha_{i-1} \in S_3 \end{cases}$$

- Each $\alpha_i = g^{a_i} y^{b_i}$, where $a_i, b_i \in \mathbb{Z}_q$ are known!
- If $\alpha_i = \alpha_j$ and $(a_i, b_i) \neq (a_j, b_j)$ then $y = g^{(a_i - a_j)(b_j - b_i)^{-1}}$.

Pollard-ρ (2/2)

- If $\alpha_i = \alpha_j$, then $\alpha_{i+1} = \alpha_{j+1}$.
- The sequence $(a_0, b_0), (a_1, b_1), \ldots$ is "essentially random".
- The Birthday bound implies that the (heuristic) expected running time is $O(\sqrt{q})$.
- We use "double runners" to reduce memory.

Index Calculus

- Let $\mathcal{B} = \{p_1, \ldots, p_B\}$ be a set of small prime **integers**.

Index Calculus

- Let $\mathcal{B} = \{p_1, \ldots, p_B\}$ be a set of small prime **integers**.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.

Index Calculus

- Let $\mathcal{B} = \{p_1, \ldots, p_B\}$ be a set of small prime **integers**.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.
  1. Choose $s_j \in \mathbb{Z}_q$ randomly and attempt to factor $g^{s_j} = \prod_i p_i^{e_{j,i}}$ as an **integer**.

Index Calculus

- Let $\mathcal{B} = \{p_1, \ldots, p_B\}$ be a set of small prime **integers**.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.
  1. Choose $s_j \in \mathbb{Z}_q$ randomly and attempt to factor $g^{s_j} = \prod_i p_i^{e_{j,i}}$ as an **integer**.
  2. If g^{s_j} factored in $\mathcal{B}$ and $e_j = (e_{j,1}, \ldots, e_{j,B})$ is linearly independent of $e_1, \ldots, e_{j-1}$, then $j \leftarrow j + 1$.

Index Calculus

- Let $\mathcal{B} = \{p_1, \ldots, p_B\}$ be a set of small prime **integers**.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.
  1. Choose $s_j \in \mathbb{Z}_q$ randomly and attempt to factor $g^{s_j} = \prod_i p_i^{e_{j,i}}$ as an **integer**.
  2. If g^{s_j} factored in $\mathcal{B}$ and $e_j = (e_{j,1}, \ldots, e_{j,B})$ is linearly independent of $e_1, \ldots, e_{j-1}$, then $j \leftarrow j + 1$.
  3. If $j < B$, then go to (1)

Index Calculus

- Let $\mathcal{B} = \{p_1, \ldots, p_B\}$ be a set of small prime **integers**.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.

Index Calculus

- Let $\mathcal{B} = \{p_1, \ldots, p_B\}$ be a set of small prime **integers**.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.
- Repeat:
  1. Choose $s \in \mathbb{Z}_q$ randomly.
  2. Attempt to factor $yg^s = \prod_i p_i^{e_i}$ as an **integer**.
  3. If a factorization is found, then output $(\sum_i a_i e_i - s) \bmod q$.

Excercise: Why doesn't this work for any cyclic group?

Example Groups

- $\mathbb{Z}_n$ additively? **Bad for crypto!**

Example Groups

- $\mathbb{Z}_n$ additively? **Bad for crypto!**
- Large prime order subgroup of $\mathbb{Z}_p^*$ with p prime. In particular $p = 2q + 1$ with q prime.

Example Groups

- $\mathbb{Z}_n$ additively? **Bad for crypto!**
- Large prime order subgroup of $\mathbb{Z}_p^*$ with p prime. In particular $p = 2q + 1$ with q prime.
- Large prime order subgroup of $\mathrm{GF}_{p^k}^*$.

Example Groups

- $\mathbb{Z}_n$ additively? **Bad for crypto!**
- Large prime order subgroup of $\mathbb{Z}_p^*$ with p prime. In particular $p = 2q + 1$ with q prime.
- Large prime order subgroup of $\mathrm{GF}_{p^k}^*$.
- "Carefully chosen" elliptic curve group.

Elliptic Curves

Groups

- We have argued that discrete logarithm problems are hard in large subgroups of $\mathbb{Z}_p^*$ and $\mathbb{F}_q^*$.
- Based on discrete logarithm problems (DL, DH, DDH) we can construct public key cryptosystems, key exchange protocols, and signature schemes.
- An elliptic curve is another candidate of a group where discrete logarithm problems are hard.

Motivation For Studying Elliptic Curves

- What if it turns out that solving discrete logarithms in $\mathbb{Z}_p^*$ is easy? Elliptic curves give an **alternative**.
- The best known DL-algorithms in an elliptic curve group with prime order q are **generic algorithms**, i.e., they have running time $O(\sqrt{q})$
- Arguably we can use **shorter keys**. This is very important in some practical applications.

Definition

Definition. A plane cubic curve E (on Weierstrass form) over a field $\mathbb{F}$ is given by a polynomial

$$y^2 = x^3 + ax + b$$

with $a, b \in \mathbb{F}$. The set of points (x, y) that satisfy this equation over $\mathbb{F}$ is written $E(\mathbb{F})$.

Definition

Definition. A plane cubic curve E (on Weierstrass form) over a field $\mathbb{F}$ is given by a polynomial

$$y^2 = x^3 + ax + b$$

with $a, b \in \mathbb{F}$. The set of points (x, y) that satisfy this equation over $\mathbb{F}$ is written $E(\mathbb{F})$.

Every plane cubic curve over a field of characteristic $\neq 2, 3$ can be written on the above form without changing any properties we care about.

Alternative Notation

We also write

$$g(x, y) = x^3 + ax + b - y^2 \quad \text{or}$$
$$y^2 = f(x)$$

where $f(x) = x^3 + ax + b$.

Singular Points

Definition. A point $(u, v) \in E(\mathbb{E})$, with $\mathbb{E}$ an extension field of $\mathbb{F}$, is **singular** if

$$\frac{\partial g(x,y)}{\partial x}(u,v) = \frac{\partial g(x,y)}{\partial y}(u,v) = 0 \ .$$

Definition. A plane cubic curve is **smooth** if $E(\overline{\mathbb{F}})$ contains no singular points[1].

[1] $\overline{\mathbb{F}}$ is the algebraic closure of $\mathbb{F}$.

What Does This Mean?

Note that

$$\frac{\partial g(x,y)}{\partial x}(x,y) = f'(x) = 3x^2 + a \quad \text{and}$$
$$\frac{\partial g(x,y)}{\partial y}(x,y) = -2y \ .$$

Thus, any singular point $(u, v) \in E(\mathbb{F})$ must have:

- $v = 0$,
- $f(u) = 0$, and $f'(u) = 0$.

Then $f(x) = (x - u)h(x)$ and $f'(x) = h(x) + (x - u)h'(x)$, so (u, v) is singular if $v = 0$ and u is a double-root of f.

Discriminant

In general a "discriminant" can be used to check if a polynomial has a double root.

Definition. The discriminant $\Delta(E)$ of a plane curve $y^2 = x^3 + ax + b$ is given by $-4a^3 - 27b^2$.

Lemma. The polynomial $f(x)$ does not have a double root iff $\Delta(E) \neq 0$, in which case the curve is called **smooth**.

Line Defined By Two Points On Curve

Let $l(x)$ be a line that intersects the curve in (u_1, v_1) and (u_2, v_2). Then

$$l(x) = k(x - u_1) + v_1$$

where

$$k = \begin{cases} \frac{v_2 - v_1}{u_2 - u_1} & \text{if } (u_1, v_1) \neq (u_2, v_2) \\ \frac{3u_1^2 + a}{2v_1} & \text{otherwise} \end{cases}$$

Line Defined By Two Points On Curve

Let $l(x)$ be a line that intersects the curve in (u_1, v_1) and (u_2, v_2). Then

$$l(x) = k(x - u_1) + v_1$$

where

$$k = \begin{cases} \frac{v_2 - v_1}{u_2 - u_1} & \text{if } (u_1, v_1) \neq (u_2, v_2) \\ \frac{3u_1^2 + a}{2v_1} & \text{otherwise} \end{cases}$$

We are cheating a little here in that we assume that we don't have $u_1 = u_2$ and $v_1 \neq v_2$ or $v_1 = v_2 = 0$. In both such cases we get a line parallel with $x = 0$ that we deal with in a special way.

Finding the Third Point

- The intersection points between $l(x)$ and the curve are given by the zeros of

$$t(x) = g(l(x), x) = f(x) - l(x)^2$$

which is a cubic polynomial with known roots u_1 and u_2.

Finding the Third Point

- The intersection points between $l(x)$ and the curve are given by the zeros of

$$t(x) = g(l(x), x) = f(x) - l(x)^2$$

which is a cubic polynomial with known roots u_1 and u_2.

- To find the third intersection point (u_3, v_3) we note that

$$t(x) = (x - u_1)(x - u_2)(x - u_3) = x^3 - (u_1 + u_2 + u_3)x^2 + r(x)$$

where $r(x)$ is linear. Thus, we can find u_3 from t's coefficients!

From Intersection Points To Group Law

- Given any two points A and B the on the curve that defines a line, we can find a third intersection point C with the curve (even if $A = B$).

From Intersection Points To Group Law

- Given any two points A and B the on the curve that defines a line, we can find a third intersection point C with the curve (even if $A = B$).
- The only exception is if our line $l(x)$ is parallel with the y-axis.

From Intersection Points To Group Law

- Given any two points A and B the on the curve that defines a line, we can find a third intersection point C with the curve (even if $A = B$).
- The only exception is if our line $l(x)$ is parallel with the y-axis.
- To "fix" this exception we add a point at infinity O, roughly at $(0, \infty)$ (the projective plane). Intuition: the sides of a long straight road seem to intersect infinitely far away.

From Intersection Points To Group Law

- We define the sum of A and B by $(x, -y)$, where (x, y) is the third intersection point of the line defined by A and B with the curve.
- We define the inverse of (x, y) by $(x, -y)$.
- The main technical difficulty in proving that this gives a group is to prove the associative law. This can be done with Bezout's theorem (not the one covered in class), or by (tedious) elementary algebraic manipulation.

Elliptic Curves

- There are many elliptic curves with special properties.
- There are many ways to represent the same curve and to implement curves as well as representing and implementing the underlying field.
- More requirements than smoothness must be satisfied for a curve to be suitable for cryptographic use.

Elliptic Curves

- There are many elliptic curves with special properties.
- There are many ways to represent the same curve and to implement curves as well as representing and implementing the underlying field.
- More requirements than smoothness must be satisfied for a curve to be suitable for cryptographic use.
- Fortunately, there are **standardized curves**.

  (I would need a **very strong** reason not to use these curves and I would be **extremely careful**, consulting researchers specializing in elliptic curve cryptography.)

Universal Hash Functions

Universal Hash Function

Definition. An ensemble $f = \{f_\alpha\}$ of hash functions $f_\alpha : X \to Y$ is (strongly) 2-universal if for every $x, x' \in X$ and $y, y' \in Y$ with $x \neq x'$ and a random α

$$\Pr[f_\alpha(x) = y \wedge f_\alpha(x') = y'] = \frac{1}{|Y|^2} \quad .$$

Universal Hash Function

Definition. An ensemble $f = \{f_\alpha\}$ of hash functions $f_\alpha : X \to Y$ is (strongly) 2-universal if for every $x, x' \in X$ and $y, y' \in Y$ with $x \neq x'$ and a random α

$$\Pr[f_\alpha(x) = y \wedge f_\alpha(x') = y'] = \frac{1}{|Y|^2} \ .$$

I.e., for any $x' \neq x$, the outputs $f_\alpha(x)$ and $f_\alpha(x')$ are uniformly and independently distributed.

In particular x and x' are both mapped to the same value with probability $1/|Y|$.

Example

Example. The function $f : \mathbb{Z}_p \to \mathbb{Z}_p$ for prime p defined by

$$f(z) = az + b \bmod p$$

is strongly 2-universal.

Proof. Let $x, x', y, y' \in \mathbb{Z}_p$ with $x \neq x'$. Then

$$\begin{pmatrix} x & 1 \\ x' & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix}$$

has a unique solution. Random (a, b) satisfies this solution with probability $\frac{1}{p^2}$.

Universal Hash Function

Universal hash functions are **not** one-way or collision resistant!

Hash Functions

Hash Function

A hash function maps arbitrarily long bit strings into bit strings of fixed length.

The output of a hash function should be “unpredictable”.

Wish List

- Finding a pre-image of an output should be hard.
- Finding two inputs giving the same output should be hard.
- The output of the function should be “random”.

etc

Ensembles of Functions (1/3)

- Let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a polynomial time computable function.

Ensembles of Functions (1/3)

- Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial time computable function.
- We can derive an ensemble $\{f_n\}_{n \in \mathbb{N}}$, with

$$f_n : \{0,1\}^n \to \{0,1\}^*$$

by setting $f_n(x) = f(x)$.

Ensembles of Functions (1/3)

- Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial time computable function.
- We can derive an ensemble $\{f_n\}_{n \in \mathbb{N}}$, with

$$f_n : \{0,1\}^n \to \{0,1\}^*$$

  by setting $f_n(x) = f(x)$.
- Note that we may recover f from the ensemble by $f(x) = f_{|x|}(x)$.

Ensembles of Functions (1/3)

- Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial time computable function.
- We can derive an ensemble $\{f_n\}_{n \in \mathbb{N}}$, with

$$f_n : \{0,1\}^n \to \{0,1\}^*$$

  by setting $f_n(x) = f(x)$.
- Note that we may recover f from the ensemble by $f(x) = f_{|x|}(x)$.
- When convenient we give definitions for a function, but it can be turned into a definition for an ensemble.

Ensembles of Functions (2/3)

- Consider $F = \{f_n\}_{n \in \mathbb{N}}$, where f_n is itself an ensemble $\{f_{n,\alpha_n}\}_{\alpha_n \in \{0,1\}^n}$, with

$$f_{n,\alpha_n} : \{0,1\}^{l(n)} \to \{0,1\}^{l'(n)}$$

  for some polynomials $l(n)$ and $l'(n)$.

Ensembles of Functions (2/3)

- Consider $F = \{f_n\}_{n \in \mathbb{N}}$, where f_n is itself an ensemble $\{f_{n,\alpha_n}\}_{\alpha_n \in \{0,1\}^n}$, with

$$f_{n,\alpha_n} : \{0,1\}^{l(n)} \to \{0,1\}^{l'(n)}$$

  for some polynomials $l(n)$ and $l'(n)$.

- Here n is the security parameter and α is a "key" that is chosen randomly.

Ensembles of Functions (2/3)

- Consider $F = \{f_n\}_{n \in \mathbb{N}}$, where f_n is itself an ensemble $\{f_{n,\alpha_n}\}_{\alpha_n \in \{0,1\}^n}$, with

$$f_{n,\alpha_n} : \{0,1\}^{l(n)} \to \{0,1\}^{l'(n)}$$

  for some polynomials $l(n)$ and $l'(n)$.

- Here n is the security parameter and α is a "key" that is chosen randomly.

- We may also view F as an ensemble $\{f_\alpha\}$, where $f_\alpha = \{f_{n,\alpha_n}\}_{n \in \mathbb{N}}$ and $\alpha = \{\alpha_n\}_{n \in \mathbb{N}}$.

Ensembles of Functions (3/3)

These conventions allow us to talk about what in everyday language is a "function" f in several convenient ways.

Now you can forget that and assume that everything works!

One-Wayness

Definition. A function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be **one-way**[2] if for every polynomial time algorithm A and a random x

$$\Pr[A(f(x)) = x' \wedge f(x') = f(x)] < \epsilon(n)$$

for a negligible function ϵ.

Normally f is computable in polynomial time in its input size.

[2] "Enkelriktad" på svenska **inte** "enväg".

Second Pre-Image Resistance

Definition. A function $h : \{0,1\}^* \to \{0,1\}^*$ is said to be **second pre-image resistant** if for every polynomial time algorithm A and a random x

$$\Pr[A(x) = x' \wedge x' \neq x \wedge f(x') = f(x)] < \epsilon(n)$$

for a negligible function ϵ.

Note that A is given not only the output of f, but also the **input** x, but it must find a **second** pre-image.

Collision Resistance

Definition. Let $f = \{f_\alpha\}_\alpha$ be an ensemble of functions. The "function" f is said to be **collision resistant** if for every polynomial time algorithm A and randomly chosen α

$$\Pr[A(\alpha) = (x, x') \wedge x \neq x' \wedge f_\alpha(x') = f_\alpha(x)] < \epsilon(n)$$

for a negligible function ϵ.

Collision Resistance

Definition. Let $f = \{f_\alpha\}_\alpha$ be an ensemble of functions. The "function" f is said to be **collision resistant** if for every polynomial time algorithm A and randomly chosen α

$$\Pr[A(\alpha) = (x, x') \wedge x \neq x' \wedge f_\alpha(x') = f_\alpha(x)] < \epsilon(n)$$

for a negligible function ϵ.

An algorithm that gets a small "advice string" for each security parameter can easily hardcode a collision for a fixed function f, which explains the random index α.

Relations for Compressing Hash Functions

- If a function is not pre-image resistant, then it is not collision-resistant.

Relations for Compressing Hash Functions

- If a function is not pre-image resistant, then it is not collision-resistant.
  1. Pick random x.
  2. Request second pre-image $x' \neq x$ with $f(x') = f(x)$.
  3. Output x' and x.

Relations for Compressing Hash Functions

- If a function is not pre-image resistant, then it is not collision-resistant.
  1. Pick random x.
  2. Request second pre-image $x' \neq x$ with $f(x') = f(x)$.
  3. Output x' and x.

- If a function is not one-way, then it is not second pre-image resistant.

Relations for Compressing Hash Functions

- If a function is not pre-image resistant, then it is not collision-resistant.
  1. Pick random x.
  2. Request second pre-image $x' \neq x$ with $f(x') = f(x)$.
  3. Output x' and x.

- If a function is not one-way, then it is not second pre-image resistant.
  1. Given random x, compute $y = f(x)$.
  2. Request pre-image x' of y.
  3. Repeat until $x' \neq x$, and output x'.

Random Oracles

Random Oracle As Hash Function

A random oracle is simply a randomly chosen function with appropriate domain and range.

A random oracle is the **perfect** hash function. Every input is mapped **independently** and **uniformly** in the range.

Let us consider how a random oracle behaves with respect to our notions of security of hash functions.

Pre-Image of Random Oracle

We assume with little loss that an adversary always "knows" if it has found a pre-image, i.e., it queries the random oracle on its output.

Theorem. Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)}(H(x)) = x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^q \quad .$$

Pre-Image of Random Oracle

We assume with little loss that an adversary always "knows" if it has found a pre-image, i.e., it queries the random oracle on its output.

Theorem. Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)}(H(x)) = x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^q \quad .$$

Proof. Each query x' satisfies $H(x') \neq H(x)$ independently with probability $1 - \frac{1}{|Y|}$.

Second Pre-Image of Random Oracle

We assume with little loss that an adversary always "knows" if it has found a second pre-image, i.e., it queries the random oracle on the input and its output.

Theorem. Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every such algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)}(x) = x' \wedge x \neq x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^{q-1} .$$

Second Pre-Image of Random Oracle

We assume with little loss that an adversary always "knows" if it has found a second pre-image, i.e., it queries the random oracle on the input and its output.

Theorem. Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every such algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)}(x) = x' \wedge x \neq x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^{q-1} .$$

Proof. Same as pre-image case, except we must waste one query on the input value to get the target in Y.

Collision Resistance of Random Oracles

We assume with little loss that an adversary always "knows" if it has found a collision, i.e., it queries the random oracle on its outputs.

Theorem. Let $H : X \to Y$ be a randomly chosen function. Then for every such algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)} = (x, x') \wedge x \neq x' \wedge H(x) = H(x')] \leq 1 - \prod_{i=1}^{q-1} \left(1 - \frac{i}{|Y|}\right)$$
$$\leq \frac{q(q-1)}{2|Y|} \quad .$$

Collision Resistance of Random Oracles

We assume with little loss that an adversary always "knows" if it has found a collision, i.e., it queries the random oracle on its outputs.

Theorem. Let $H : X \to Y$ be a randomly chosen function. Then for every such algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)} = (x, x') \wedge x \neq x' \wedge H(x) = H(x')] \leq 1 - \prod_{i=1}^{q-1} \left(1 - \frac{i}{|Y|}\right) \leq \frac{q(q-1)}{2|Y|} .$$

Proof. $1 - \frac{i-1}{|Y|}$ bounds the probability that the ith query does not give a collision for any of the $i-1$ previous queries, conditioned on no previous collision.

Iterated Hash Functions

Merkle-Damgård (1/3)

Suppose that we are given a collision resistant hash function

$$f : \{0,1\}^{n+t} \to \{0,1\}^n \ .$$

How can we construct a collision resistant hash function

$$h : \{0,1\}^* \to \{0,1\}^n$$

mapping any length inputs?

Merkle-Damgård (2/3)

Construction.

1. Let $x = (x_1, \ldots, x_k)$ with $|x_i| = t$ and $0 < |x_k| \leq t$.
2. Let x_{k+1} be the total number of bits in x.
3. Pad x_k with zeros until it has length t.
4. $y_0 = 0^n$, $y_i = f(y_{i-1}, x_i)$ for $i = 1, \ldots, k+1$.
5. Output y_{k+1}

Here the total number of bits is bounded by $2^t - 1$, but this can be relaxed.

Merkle-Damgård (3/3)

Suppose A finds collisions in Merkle-Damgård.

- If the number of bits differ in a collision, then we can derive a collision from the last invocation of f.
- If not, then we move backwards until we get a collision. Since both inputs have the same length, we are guaranteed to find a collision.

Standardized Hash Functions

Standardized Hash Functions

Despite that theory says it is impossible, in practice people simply live with **fixed** hash functions and use them as if they are randomly chosen functions.

SHA

- Secure Hash Algorithm (SHA-0,1, and the SHA-2 family) are hash functions standardized by NIST to be used in, e.g., signature schemes and random number generation.
- SHA-0 was **weak** and withdrawn by NIST. SHA-1 was **withdrawn** 2010. SHA-2 family is based on similar ideas but seems safe so far...
- All are **iterated** hash functions, starting from a basic **compression function**.

SHA-3

- NIST ran an open competition for the next hash function, named SHA-3. Several groups of famous researchers submitted proposals.
- Call for SHA-3 explicitly asked for "different" hash functions.
- It might be a good idea to read about SHA-1 for comparison.
- The competition ended October 2, 2012, and the hash function **Keccak was selected as the winner**.
- This was constructed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche,

MACs

Message Authentication Code

- Message Authentication Codes (MACs) are used to ensure integrity and authenticity of messages.

Message Authentication Code

- Message Authentication Codes (MACs) are used to ensure integrity and authenticity of messages.
- Scenario:
  1. Alice and Bob share a common key k.
  2. Alice computes an authentication tag $\alpha = \mathsf{MAC}_\mathsf{k}(m)$ and sends (m, α) to Bob.
  3. Bob receives (m', α') from Alice, but before accepting m' as coming from Alice, Bob checks that $\mathsf{MAC}_\mathsf{k}(m') = \alpha'$.

Security of a MAC

Definition. A message authentication code MAC is secure if for a random key k and every polynomial time algorithm A,

$$\Pr[A^{\mathsf{MAC}_k(\cdot)} = (m, \alpha) \wedge \mathsf{MAC}_k(m) = \alpha \wedge \forall i : m \neq m_i]$$

is negligible, where m_i is the ith query to the oracle $\mathsf{MAC}_k(\cdot)$.

Random Oracle As MAC

- Suppose that $H : \{0,1\}^* \to \{0,1\}^n$ is a random oracle.
- Then we can construct a MAC as $\mathrm{MAC}_k(m) = H(k, m)$.

Could we plug in an iterated hash function in place of the random oracle?

HMAC

- Let $H : \{0,1\}^* \to \{0,1\}^n$ be a "cryptographic hashfunction", e.g., SHA-256.
- $\mathrm{HMAC}_{k_1,k_2}(x) = H\big(k_2 \| H(k_1 \| x)\big)$
- This is provably secure under the assumption that
  - $H(k_1 \| \cdot)$ is unknown-key collision resistant, and
  - $H(k_2 \| \cdot)$ is a secure MAC.

CBC-MAC

Let E be a secure block-cipher, and $x = (x_1, \ldots, x_t)$ an input. The MAC-key is simply the block-cipher key.

1. $y_0 = 000 \ldots 0$
2. For $i = 1, \ldots, t$, $y_i = \mathsf{E}_k(y_{i-1} \oplus x_i)$
3. Return y_t.

Is this secure?

Universal Hashfunction As MAC

Theorem. A t-universal hashfunction f_α for a randomly chosen secret α is an **unconditionally secure** MAC, provided that the number queries is smaller than t.

Signature Schemes

Digital Signature

- A digital signature is the **public-key** equivalent of a MAC; the receiver verifies the integrity and authenticity of a message.
- Does a digital signature replace a real handwritten one?

Textbook RSA Signature (1/2)

- Generate RSA keys $((N, e), (p, q, d))$.
- To sign a message $m \in \mathbb{Z}_N$, compute $\sigma = m^d \bmod N$.
- To verify a signature σ of a message m, verify that $\sigma^e = m \bmod N$.

Textbook RSA Signature (2/2)

- Are Textbook RSA Signatures any good?

Textbook RSA Signature (2/2)

- Are Textbook RSA Signatures any good?
- If σ is a signature of m, then $\sigma^2 \bmod N$ is a signature of $m^2 \bmod N$.

Textbook RSA Signature (2/2)

- Are Textbook RSA Signatures any good?
- If σ is a signature of m, then $\sigma^2 \bmod N$ is a signature of $m^2 \bmod N$.
- If σ_1 and σ_2 are signatures of m_1 and m_2, then $\sigma_1\sigma_2 \bmod N$ is a signature of $m_1 m_2 \bmod N$

Textbook RSA Signature (2/2)

- Are Textbook RSA Signatures any good?
- If σ is a signature of m, then $\sigma^2 \bmod N$ is a signature of $m^2 \bmod N$.
- If σ_1 and σ_2 are signatures of m_1 and m_2, then $\sigma_1\sigma_2 \bmod N$ is a signature of $m_1 m_2 \bmod N$
- We can also pick a signature σ and compute the message it is a signature of by $m = \sigma^e \bmod N$.

Textbook RSA Signature (2/2)

- Are Textbook RSA Signatures any good?
- If σ is a signature of m, then $\sigma^2 \bmod N$ is a signature of $m^2 \bmod N$.
- If σ_1 and σ_2 are signatures of m_1 and m_2, then $\sigma_1\sigma_2 \bmod N$ is a signature of $m_1 m_2 \bmod N$
- We can also pick a signature σ and compute the message it is a signature of by $m = \sigma^e \bmod N$.

We must be more careful!

Signature Scheme

- Gen **generates a key pair** $(\mathsf{pk}, \mathsf{sk})$.
- Sig takes a secret key sk and a message m and **computes a signature** σ.
- Vf takes a public key pk, a message m, and a candidate signature σ, **verifies the candidate signature**, and outputs a single-bit verdict.

Existential Unforgeability

Definition. A signature scheme $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Vf})$ is **secure against existential forgeries** if for every polynomial time algorithm and a random key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$,

$$\Pr\left[A^{\mathsf{Sig}_{\mathsf{sk}}(\cdot)}(\mathsf{pk}) = (m, \sigma) \wedge \mathsf{Vf}_{\mathsf{pk}}(m, \sigma) = 1 \wedge \forall i : m \neq m_i\right]$$

is negligible where m_i is the ith query to $\mathsf{Sig}_{\mathsf{sk}}(\cdot)$.

Provably Secure Signature Schemes

Provably secure signature schemes exists if one-way functions exist (in plain model without ROM), but the construction is more involved and typically less efficient.

Provably secure signature schemes are rarely used in practice!

Standards used in practice: RSA Full Domain Hash, DSA, EC-DSA. The latter two may be viewed as variants of Schnorr signatures.

Based on Trapdoor One-Way Permutations

Trapdoor One-Way Permutations

Let $f = \{f_\alpha\}$ be an ensemble of **permutations** (bijections).

- Gen **generates a random key pair** $\alpha = (\mathsf{pk}, \mathsf{sk})$.
- Eval takes pk and x as input and **efficiently evaluates** $f_\alpha(x)$.
- Invert takes sk and y as input and **efficiently evaluates the inverse** $f_\alpha^{-1}(y)$.

One-way if $\mathsf{Eval}_{\mathsf{pk}}(\cdot)$ is one-way for a random pk.

Trapdoor One-Way Permutations

Let $f = \{f_\alpha\}$ be an ensemble of **permutations** (bijections).

- Gen **generates a random key pair** $\alpha = (\mathsf{pk}, \mathsf{sk})$.
- Eval takes pk and x as input and **efficiently evaluates** $f_\alpha(x)$.
- Invert takes sk and y as input and **efficiently evaluates the inverse** $f_\alpha^{-1}(y)$.

One-way if $\mathsf{Eval}_{\mathsf{pk}}(\cdot)$ is one-way for a random pk.

RSA is a trap-door permutation over $\mathbb{Z}_N^*$.

Trapdoor One-Way Permutations (Less Formal)

Let $f = \{f_\alpha\}$ be an ensemble of **permutations** (bijections).

- Gen **generates a pair** $(f_\alpha, f_\alpha^{-1})$.
- Eval takes pk and x as input and **efficiently evaluates** $f_\alpha(x)$.
- Invert takes sk and y as input and **efficiently evaluates the inverse** $f_\alpha^{-1}(y)$.

One-way if f_α is one-way when chosen randomly.

RSA is a trap-door permutation over $\mathbb{Z}_N^*$.

Full Domain Hash Signature In ROM

Let $f = \{f_\alpha\}$ be a trapdoor permutation (family) and let $H : \{0,1\}^* \to \{0,1\}^n$ be a random oracle.

- Gen samples a pair $(f_\alpha, f_\alpha^{-1})$.
- Sig takes f_α^{-1} and a message m as input and outputs $f_\alpha^{-1}\big(H(m)\big)$.
- Vf takes f_α, a message m, and a candidate signature σ as input, and outputs 1 if $f_\alpha(\sigma) = H(m)$ and 0 otherwise.

Based on Proofs of Knowledge

Proof of Knowledge of Exponent

In an **identification scheme** one party convinces another that it holds some special token.

Proof of Knowledge of Exponent

In an **identification scheme** one party convinces another that it holds some special token.

- Let G_q be a group of prime order q with generator g.

Proof of Knowledge of Exponent

In an **identification scheme** one party convinces another that it holds some special token.

- Let G_q be a group of prime order q with generator g.
- Let $x \in \mathbb{Z}_q$ and define $y = g^x$.

Proof of Knowledge of Exponent

In an **identification scheme** one party convinces another that it holds some special token.

- Let G_q be a group of prime order q with generator g.
- Let $x \in \mathbb{Z}_q$ and define $y = g^x$.
- Can we prove knowledge of x without disclosing anything about x?

Schnorr's Protocol (1/3)

1. The prover chooses $r \in \mathbb{Z}_q$ randomly and hands $\alpha = g^r$ to the verifier.

Schnorr's Protocol (1/3)

1. The prover chooses $r \in \mathbb{Z}_q$ randomly and hands $\alpha = g^r$ to the verifier.

2. The verifier chooses $c \in \mathbb{Z}_q$ randomly and hands it to the prover.

Schnorr's Protocol (1/3)

1. The prover chooses $r \in \mathbb{Z}_q$ randomly and hands $\alpha = g^r$ to the verifier.

2. The verifier chooses $c \in \mathbb{Z}_q$ randomly and hands it to the prover.

3. The prover computes $d = cx + r \bmod q$ and hands d to the verifier.

Schnorr's Protocol (1/3)

1. The prover chooses $r \in \mathbb{Z}_q$ randomly and hands $\alpha = g^r$ to the verifier.

2. The verifier chooses $c \in \mathbb{Z}_q$ randomly and hands it to the prover.

3. The prover computes $d = cx + r \bmod q$ and hands d to the verifier.

4. The verifier accepts if $y^c \alpha = g^d$.

Schnorr's Protocol (1/3)

1. The prover chooses $r \in \mathbb{Z}_q$ randomly and hands $\alpha = g^r$ to the verifier.
2. The verifier chooses $c \in \mathbb{Z}_q$ randomly and hands it to the prover.
3. The prover computes $d = cx + r \bmod q$ and hands d to the verifier.
4. The verifier accepts if $y^c \alpha = g^d$.

Suppose that a machine convinces us in the protocol with probability δ. Does it mean that it knows x such that $y = g^x$?

Schnorr's Protocol (2/3)

Schnorr's Protocol (2/3)

1. Run the machine to get α.

Schnorr's Protocol (2/3)

1. Run the machine to get α.

2. Complete the interaction twice using **the same** α, once for a challenge c and once for a challenge c', where $c, c' \in \mathbb{Z}_q$ are chosen randomly.

Schnorr's Protocol (2/3)

1. Run the machine to get α.

2. Complete the interaction twice using **the same** α, once for a challenge c and once for a challenge c', where $c, c' \in \mathbb{Z}_q$ are chosen randomly.

3. Repeat from (1) until the resulting interactions (α, c, d) and (α, c', d') are accepting and $c \neq c'$.

Schnorr's Protocol (2/3)

1. Run the machine to get α.

2. Complete the interaction twice using **the same** α, once for a challenge c and once for a challenge c', where $c, c' \in \mathbb{Z}_q$ are chosen randomly.

3. Repeat from (1) until the resulting interactions (α, c, d) and (α, c', d') are accepting and $c \neq c'$.

4. Note that:

$$y^{c-c'} = \frac{y^c}{y^{c'}} = \frac{y^c \alpha}{y^{c'} \alpha} = \frac{g^d}{g^{d'}} = g^{d-d'}$$

which gives the logarithm $x = (d - d')(c - c')^{-1} \bmod q$ such that $y = g^x$.

Schnorr's Protocol (3/3)

- Anybody can sample $c, d \in \mathbb{Z}_q$ randomly and compute $\alpha = g^d / y^c$.
- The resulting tuple (α, c, d) has **exactly** the same distribution as the transcript of an interaction!

Such protocols are called (honest verifier) **zero-knowledge proofs of knowledge**.

Schnorr's Signature Scheme In ROM

Let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a random oracle.

- Gen chooses $x \in \mathbb{Z}_q$ randomly, computes $y = g^x$ and outputs $(\mathsf{pk}, \mathsf{sk}) = (y, x)$.
- Sig does the following on input x and m:
  1. it chooses $r \in \mathbb{Z}_q$ randomly and computes $\alpha = g^r$,
  2. it computes $c = H(y, \alpha, m)$,
  3. it computes $d = cx + r \bmod q$ and outputs (α, d).
- Vf takes the public key y, a message m, and a candidate signature (α, d), and accepts iff $y^{H(y,\alpha,m)}\alpha = g^d$.

PKIs

Problem

- We have constructed public-key cryptosystems and signature schemes.
- Only the holder of the secret key can decrypt ciphertexts and sign messages.
- How do we **know** who holds the secret key corresponding to a public key?

Signing Public Keys of Others

- Suppose that Alice computes a signature $\sigma_{A,B} = \mathsf{Sig}_{\mathsf{sk}_A}(\mathsf{pk}_B, \mathit{Bob})$ of Bob's public key pk_B and his identity and hands it to Bob.
- Suppose that Eve holds Alice's public key pk_A.
- Then **anybody** can hand $(\mathsf{pk}_B, \sigma_{A,B})$ **directly** to Eve, and Eve will be convinced that pk_B is Bob's key (assuming she trusts Alice).

Certificate

- A **certificate** is a signature of a public key along with some information on how the key may be used, e.g., it may allow the holder to issue certificates.
- A certificate is valid for a given setting if the signature is valid and the usage information in the certificate matches that of the setting.
- Some parties must be trusted to issue certificates. These parties are called Certificate Authorities (CA).

Certificate Chains

A CA may be "distributed" using in certificate chains.

- Suppose that Bob holds valid certificates

$$\sigma_{0,1}, \sigma_{1,2}, \ldots, \sigma_{n-1,n}$$

  where $\sigma_{i-1,i}$ is a certificate of pk_{P_i} by P_{i-1}.

- Who does Bob trust?

Pseudo-random Generators

Randomness

- Everything we have done so far requires randomness!

Randomness

- Everything we have done so far requires randomness!
- Can we "generate" random strings?

Physical Randomness and Deterministic Algorithms

- We could flip actual coins. This would be extremely impractical and slow (and booring unless you are Rain man).

Physical Randomness and Deterministic Algorithms

- We could flip actual coins. This would be extremely impractical and slow (and booring unless you are Rain man).
- We could generate "physical" randomness using hardware, e.g., measuring radioactive decay
  - Slow or expensive.
  - Hard to verify and trust.
  - Biased output.

Physical Randomness and Deterministic Algorithms

- We could flip actual coins. This would be extremely impractical and slow (and booring unless you are Rain man).
- We could generate "physical" randomness using hardware, e.g., measuring radioactive decay
  - Slow or expensive.
  - Hard to verify and trust.
  - Biased output.
- We could use a deterministic algorithm that outputs a "random looking string", but would that be secure?

Pseudo-Random Generator

A pseudo-random generator requires a **short random string** and deterministically expands this to a **longer "random looking" string**.

Pseudo-Random Generator

A pseudo-random generator requires a **short random string** and deterministically expands this to a **longer "random looking" string**.

This looks promising:

- Fast and cheap?
- Practical since it can be implemented in software or hardware?
- What is "random looking"?

Pseudo-Random Generator

Definition. An efficient algorithm PRG is a **pseudo-random generator (PRG)** if there exists a polynomial $p(n) > n$ such that for every polynomial time adversary A, if a seed $s \in \{0,1\}^n$ and a random string $u \in \{0,1\}^{p(n)}$ are chosen randomly, then

$$|\Pr[A(\text{PRG}(s)) = 1] - \Pr[A(u) = 1]|$$

is negligible.

Informally, A can not distinguish $\text{PRG}(s)$ from a truly random string in $\{0,1\}^{p(n)}$.

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

- Suppose that there exists a PRG that extends its output by a **single bit**.

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

- Suppose that there exists a PRG that extends its output by a **single bit**.
- This would **not be very useful** to us, e.g., to generate a random prime we need many random bits.

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

- Suppose that there exists a PRG that extends its output by a **single bit**.
- This would **not be very useful** to us, e.g., to generate a random prime we need many random bits.
- Can we use the given PRG to construct another PRG which extends its output more?

Increasing Extension (2/2)

Construction. Let PRG be a pseudo-random generator. We let PRG_t be the algorithm that takes $s_{-1} \in \{0,1\}^n$ as input, computes $s_0, s_2, \ldots, s_{t-1}$ and $b_0, \ldots, b_{t-1}$ as

$$(s_i, b_i) = \mathrm{PRG}(s_{i-1})$$

and outputs $(b_0, \ldots, b_{t-1})$.

Increasing Extension (2/2)

Construction. Let PRG be a pseudo-random generator. We let PRG_t be the algorithm that takes $s_{-1} \in \{0,1\}^n$ as input, computes $s_0, s_2, \ldots, s_{t-1}$ and $b_0, \ldots, b_{t-1}$ as

$$(s_i, b_i) = \mathrm{PRG}(s_{i-1})$$

and outputs $(b_0, \ldots, b_{t-1})$.

Theorem. Let $p(n)$ be a polynomial and PRG a pseudo-random generator. Then $\mathrm{PRG}_{p(n)}$ is a pseudo-random generator that on input $s \in \{0,1\}^n$ outputs a string in $\{0,1\}^{p(n)}$.

Increasing Extension (2/2)

Construction. Let PRG be a pseudo-random generator. We let PRG_t be the algorithm that takes $s_{-1} \in \{0,1\}^n$ as input, computes $s_0, s_2, \ldots, s_{t-1}$ and $b_0, \ldots, b_{t-1}$ as

$$(s_i, b_i) = \mathrm{PRG}(s_{i-1})$$

and outputs $(b_0, \ldots, b_{t-1})$.

Theorem. Let $p(n)$ be a polynomial and PRG a pseudo-random generator. Then $\mathrm{PRG}_{p(n)}$ is a pseudo-random generator that on input $s \in \{0,1\}^n$ outputs a string in $\{0,1\}^{p(n)}$.

We can go on "forever"!

Random String From Random Oracle

Theorem. If $F : \{0,1\}^n \to \{0,1\}^m$ is a random function, then $(F(0), F(1), F(2), \ldots, F(t-1))$ is a tm-bit string.

Random String From Random Oracle

Theorem. If $F : \{0,1\}^n \to \{0,1\}^m$ is a random function, then $(F(0), F(1), F(2), \ldots, F(t-1))$ is a tm-bit string.

Can we do this using a pseudo-random function?

Can we replace the random function by SHA-2?

Pseudo-Random Function

Recall the definition of a pseudo-random function.

Definition. A family of functions $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is pseudo-random if for all polynomial time oracle adversaries A

$$\left| \Pr_K \left[A^{F_K(\cdot)} = 1 \right] - \Pr_{R:\{0,1\}^n \to \{0,1\}^n} \left[A^{R(\cdot)} = 1 \right] \right|$$

is negligible.

Pseudo-Random Generator From Pseudo-Random Function

Theorem. Let $\{F_K\}_{K \in \{0,1\}^k}$ be a pseudo-random function for a random choice of K. Then the PRG defined by:

$$\mathrm{PRG}(s) = (F_s(0), F_s(1), F_s(2), \ldots, F_s(t))$$

is a pseudo-random generator.

Pseudo-Random Function From Pseudo-Random Generator

Construction. Let $\mathrm{PRG} : \{0,1\}^k \to \{0,1\}^{2k}$ be a pseudo-random generator, and define a family of functions $F = \{F_K\}_{K \in \{0,1\}^k}$ as follows.

Let $x_{[i]} = (x_0, \ldots, x_i)$.

On key K and input x, F_K computes its output as follows:

1. Computes $(r_0^0, r_1^0) = \mathrm{PRG}(K)$.
2. Computes
$$(r^i_{x_{[i-1]}\|0}, r^i_{x_{[i-1]}\|1}) = \mathrm{PRG}(r^{i-1}_{x_{[i-1]}})$$
for $i = 1, \ldots, n-1$.
3. Outputs $r_{x_{[n-1]}}$.

One-Way Permutation

Definition. A family $F = \{f_n\}$ of **permutations** $f_n : \{0,1\}^n \to \{0,1\}^n$ is said to be **one-way** if for a random $x \in \{0,1\}^n$ and every polynomial time algorithm A

$$\Pr[A(f_n(x)) = x] < \epsilon(n)$$

for a negligible function $\epsilon(n)$.

(Note that for permutations we may request the unique preimage.)

Hardcore Bit

Definition. Let $F = \{f_n\}$ be a family of permutations $f_n : \{0,1\}^n \to \{0,1\}^n$ and $B = \{b_n\}$ a family of "bits" $b_n : \{0,1\}^n \to \{0,1\}$. Then B is a **hardcore bit** of F if for random $x \in \{0,1\}^n$ and every polynomial time algorithm A

$$|\Pr[A(f_n(x)) = b_n(x)] - 1/2| < \epsilon(n)$$

for a negligible function $\epsilon(n)$.

Theorem. For every one-way permutation, there is a (possibly different) one-way permutation with a hardcore bit.

PRG from One-Way Permutation

Construction. Let F be a one-way permutation and define PRG by $\mathrm{PRG}_n(s) = f_n(s) \| b_n(s)$ for $x \in \{0,1\}^n$.

PRG from One-Way Permutation

Construction. Let F be a one-way permutation and define PRG by $\mathrm{PRG}_n(s) = f_n(s) \| b_n(s)$ for $x \in \{0,1\}^n$.

Theorem. $\mathrm{PRG} : \{0,1\}^n \to \{0,1\}^{n+1}$ is a pseudo-random generator.

PRG From Any One-Way Function

Theorem. There is a construction PRG_f that is a PRG if f is a one-way function (possibly non-permutation).

The construction is very involved and is completely impractical.

What Is Used In Practice?

Various standards contain some of the following elements.

- Fast hardware generator + "algorithmic strengthening".
- `/dev/random`
  - Entropy gathering deamon with estimate of amount of entropy.
  - FreeBSD: Executes the PRG *Yarrow* (or *Futura*) pseudo-random algorithm.
  - SunOS and Un*xes use similar approaches.
  - Windows has similar devices.
- Stream cipher, e.g. block-cipher in CFB or CTR mode.
- Hashfunction with secret prefix and counter (essentially our PRF$\rightarrow$PRG construction).

Infamous Mistakes

- (1995) The original Netscape SSL code used time of the day and process IDs to seed its pseudorandom giving **way too little** entropy in the seed.
- (2008) Debian's OpenSSL commented out a critical part of the code that reduced the entropy of keys drastically!
- (2012) RSA public keys with common factors.

Important Conclusions

- Security bugs are **not** found by testing!
- With an insecure pseudo-random generator anything on top of it will be insecure.
- Any critical code must be reviewed after every modification, e.g, keep hashes of critical code.