Lecture Notes

APPLIED CRYPTOGRAPHY AND DATA SECURITY

Dr. Christof Paar

Cryptography and Information Security (CRIS) Group

Department of Electrical & Computer Engineering

Worcester Polytechnic Institute

Worcester, MA 01609

http://www.ece.wpi.edu/Research/crypt

Preface

These lecture notes are not meant as a replacement of a more comprehensive textbook. Rather, the notes at hand present the essentials of modern applied cryptography in compact form and should accompany the lecture in conjunction with one of the books mentioned below. The notes grew out of an introductory graduate course in cryptography which I have taught twelve times by now at Worcester Polytechnic Institute and in industry. Remarks, questions, and classroom discussions by our graduate students as well as by the staff of GTE Governments Systems, MA, and Philips Research, NY, greatly helped to improve the lecture notes.

I tried to present modern cryptography in a way that is accessible for engineers without any background in abstract mathematics. There is a focus on private-key and public-key algorithms, an understanding of which appears to be extremely helpful for the development of real-world applications. However, protocol-related issues such as security services, key distributions, and identification are also treated.

The lecture notes work well together with an actual book. I've used Doug Stinson's excellent textbook, [Sti95], as well as Bruce Schneier's comprehensive compilation, [Sch93]. The treatment of topics in these lecture notes loosely follow the presentation in Stinson's book. For those interested in an in-depth understanding of the field, including many theoretical topics, the handbook by Alfred Menezes, Paul van Oorschot, and Scott Vanstone, [AM97], can be strongly recommended for additional reading. Another good book which is more introductory is William Stalling's recent text book [Sta99].

I would like to express my deep gratitude to my graduate students Jorge Guajardo and Martin Rosner, who were in charge of typing the notes and of drawing all figures and tables. Their many suggestions and proof reading greatly improved the notes.

Christof Paar

May 2000

Table of Contents

Chapter 1

Introduction to Cryptography and Data Security

1.1 Literature Recommendations

Course Textbooks: [Sti95] or [Sch93].

Further Reading - the following books are excellent supplements to the course textbook:

1. [AM97] - great compilation of theoretical and practical aspects of many crypto schemes. Unique since it includes many theoretical topics that are hard to find otherwise. Highly recommended.

2. [Sta95] - Very readable treatment of algorithms and standards relevant to cryptography in networks.

1.2 Overview

Brief History of Cryptography

- Private-Key: all encryption and decryption schemes dating from BC to 1976.

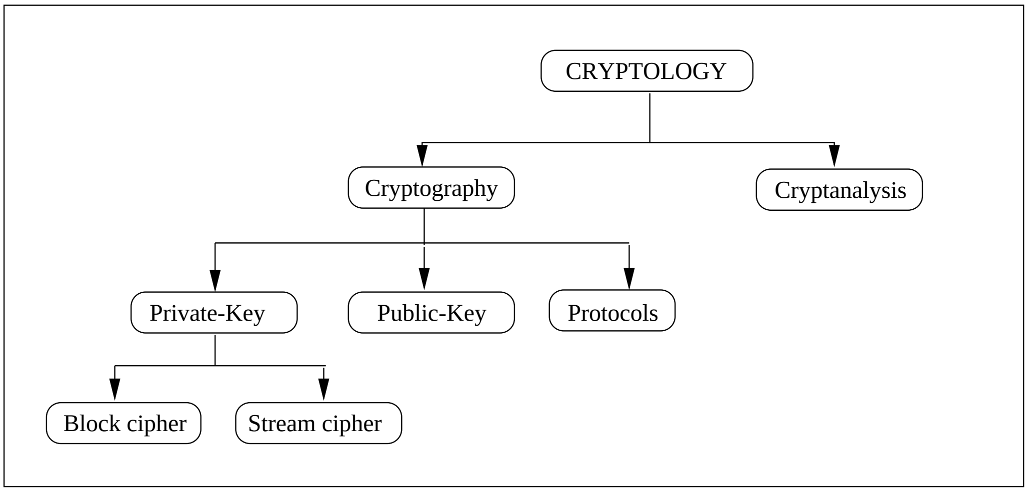


Figure 1.1: Overview on the field of cryptology

- Public-Key: in 1976 the first public-key scheme was introduced by Diffie-Hellman key exchange protocol.

- Hybrid Approach: in today's protocol, very often hybrid schemes are applied which use private and public-key algorithms.

1.3 Private-Key Cryptosystems

Sometimes these schemes are also referred to as *symmetric*, *single-key*, and *secret-key* approaches.

Problem Statement: Alice and Bob want to communication over an un-secure channel (e.g., computer network, satellite link). They want to prevent Oscar (the bad guy) from listening.

Solution: Use of private-key cryptosystems (these have been around since BC) such that if Oscar reads the encrypted version y of the message x over the un-secure channel, he will not be able to understand its content because x is what really was sent.

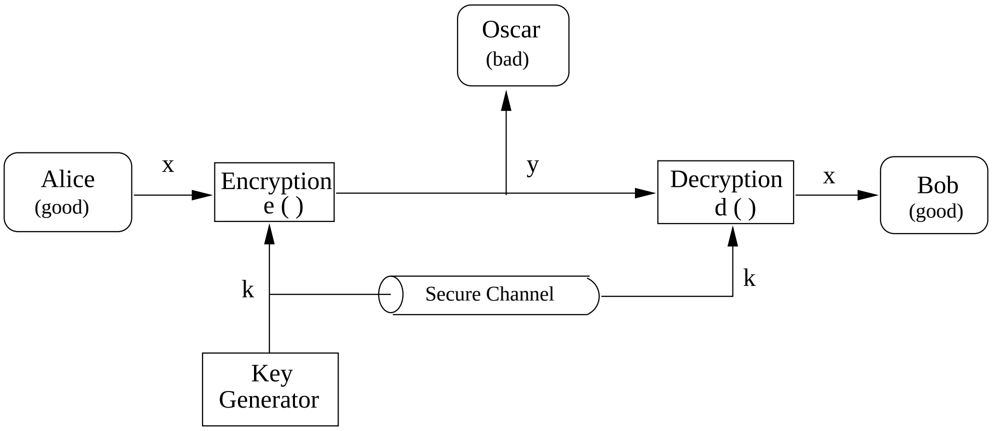


Figure 1.2: Private-key cryptosystem

Some important definitions:

1a)	x is called the "plaintext"
1b)	$\mathcal{P}= \{x_1, x_2, \ldots, x_p\}$ is the (finite) "plaintext space"
2a)	y is called the "ciphertext"
2b)	$\mathcal{C}= \{y_1, y_2, \ldots, y_c\}$ is the (finite) "ciphertext space"
3a)	k is called the "key"
3b)	$\mathcal{K}= \{k_1, k_2, \ldots, k_l\}$ is the finite "key space"
4a)	There are l encryption functions $e_{k_i} : \mathcal{P} \rightarrow \mathcal{C}$ (or: $e_{k_i}(x) = y$)
4b)	There are l decryption functions $d_{k_i} : \mathcal{C} \rightarrow \mathcal{P}$ (or: $d_{k_i}(y) = x$)
4c)	e_{k_1} and d_{k_2} are inverse functions if $k_1 = k_2 : d_{k_i}(y) = d_{k_i}(e_{k_i}(x)) = x$ for all $k_i \in \mathcal{K}$

Example: Data Encryption Standard (DES)

- $\mathcal{P} = \mathcal{C}= \{0, 1, 2, \ldots, 2^{64} - 1\}$ (each x_i has 64 bits: $x_i = 010\ldots0110$)
- $\mathcal{K}= \{0, 1, 2, \ldots, 2^{56} - 1\}$ (each k_i has 56 bits)
- encryption (e_k) and decryption (d_k) will be described in Chapter 4

1.4 Cryptanalysis

Definition: The science of recovering the plaintext x from the ciphertext y without the knowledge of the key (Oscar's job).

Rules of the game:

The cryptanalysis rules are known as *Kerckhoff's Principle*:

1. Oscar knows the cryptosystem (encryption and decryption algorithms).
2. Oscar does not know the key.

1.4.1 Attacks against Cryptoalgorithms

1. Ciphertext-only attack

Oscar's knowledge: some $y_1 = e_k(x_1)$, $y_2 = e_k(x_2)$, ...
Oscar's goal : obtain $x_1, x_2, \ldots$ or the key k.

2. Known plaintext attack

Oscar's knowledge: some pairs $(x_1, y_1 = e_k(x_1)), (x_2, y_2 = e_k(x_2)) \ldots$
Oscar's goal : obtain the key k.

3. Chosen plaintext attack

Oscar's knowledge: some pairs $(x_1, y_1 = e_k(x_1)), (x_2, y_2 = e_k(x_2)) \ldots$ of which he can choose $x_1, x_2, \ldots$
Oscar's goal : obtain the key k.

4. Chosen ciphertext attack

Oscar's knowledge: some pairs $(x_1, y_1 = e_k(x_1)), (x_2, y_2 = e_k(x_2)) \ldots$ of which he can choose

$y_1, y_2, \ldots$

Oscar's goal : obtain the key k.

1.5 Some Number Theory

Modulo operation:

Question: What is $12 \bmod 9$?

Answer: $12 \bmod 9 \equiv 3$

or $12 \equiv 3 \bmod 9$.

Definition 1.5.1 Modulo Operation

Let $a, r, m \in Z$ (where Z is a set of all integers) and $m > 0$. We write $a \equiv r \bmod m$ if m divides $r - a$.

"m" is called the modulus.

"r" is called the remainder.

Some remarks on the modulo operation:

- How is the remainder computed?

  It is always possible to write $a \in Z$, such that

  $a = q \cdot m + r;\ 0 \leq r < m$

  Now since $a - r = q \cdot m$ (m divides $a - r$) and $a \equiv r \bmod m$.

  Note that $r \in \{0, 1, 2, \ldots, m - 1\}$.

  Example:

  $a = 42;\ m = 9$

  $42 = 4 \cdot 9 + 6$ therefore $42 \equiv 6 \bmod 9$.

- C programming command : "%" (C can return a negative value)

  `r = 42 % 9` returns r = 6

  but `r = -42 % 9` returns r = -6 $\rightarrow$ if remainder is negative, add modulus m:

  $-6 + 9 = 3 \equiv -42 \mod 9$

Ring:

Definition 1.5.2 *The "ring Z_m" consists of:*

1. *The set $Z_m = \{0, 1, 2, \ldots, m-1\}$*
2. *Two operations "+" and "$\times$" for all $a, b \in Z_m$ such that:*
   - $a + b \equiv c \mod m$ $(c \in Z_m)$
   - $a \times b \equiv d \mod m$ $(d \in Z_m)$

Example: $m = 9$

$Z_9 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

$6 + 8 = 14 \equiv 5 \mod 9$

$6 \times 8 = 48 \equiv 3 \mod 9$

Definition 1.5.3 Some important properties of the ring $Z_m = \{0, 1, 2, \ldots, m-1\}$

1. *The* additive identity *is the element zero "0": $a + 0 = a \bmod m$, for any $a \in Z_m$.*
2. *The* additive inverse *"$-a$" of "a" is such that $a+(-a) \equiv 0 \bmod m$: $-a = m-a$, for any $a \in Z_m$.*
3. *Addition is* closed*: i.e., for any $a, b \in Z_m$, $a + b \in Z_m$.*
4. *Addition is* commutative*: i.e., for any $a, b \in Z_m$, $a + b = b + a$.*
5. *Addition is* associative*: i.e., for any $a, b \in Z_m$, $(a + b) + c = a + (b + c)$.*
6. *The* multiplicative identity *is the element one "1": $a \times 1 \equiv a \bmod m$, for any $a \in Z_m$.*
7. *The* multiplicative inverse *"a^{-1}" of "a" is such that $a \times a^{-1} = 1 \bmod m$: An element a has a multiplicative inverse "a^{-1}" if and only if $\gcd(a, m) = 1$.*
8. *Multiplication is* closed*: i.e., for any $a, b \in Z_m$, $ab \in Z_m$.*
9. *Multiplication is* commutative*: i.e., for any $a, b \in Z_m$, $ab = ba$.*
10. *Multiplication is* associative*: i.e., for any $a, b \in Z_m$, $(ab)c = a(bc)$.*

Some remarks on the ring Z_m:

- Roughly speaking, a ring is a structure in which we can add, subtract, multiply, and sometimes divide.

- **Definition 1.5.4** *If* $\gcd(a, m) = 1$*, then* a *and* m *are* "relatively prime" *and the multiplicative inverse of* a *exists.*

  Example:

  i) **Question**: does multiplicative inverse exist with $15 \bmod 26$?

  Answer: yes — $gcd(15, 26) = 1$

  ii) **Question**: does multiplicative inverse exist with $14 \bmod 26$?

  Answer: no — $gcd(14, 26) \neq 1$

- The modulo operation can be applied whenever we want:

  $(a + b) \bmod m = [(a \bmod m) + (b \bmod m)] \bmod m.$

  $(a \times b) \bmod m = [(a \bmod m) \times (b \bmod m)] \bmod m.$

  Example: $3^8 \bmod 7 = ?$

  i) $3^8 = 3^4 \cdot 3^4 = (81 \bmod 7) \cdot (81 \bmod 7) \equiv 4 \cdot 4 = 16 \equiv 2 \bmod 7.$

  ii) $3^8 = 6561 \equiv 2 \bmod 7$, since $6561 = 937 \cdot 7 + 2$.

  As we see, it is almost always of computational advantage to apply the modulo reduction as soon as we can.

- The ring Z_m, and thus the integer arithmetic with the modulo operation, is of central importance to modern public-key cryptography. In practice, the integers are represented with 150–2048 bits.

1.6 Simple Blockciphers

Recall:

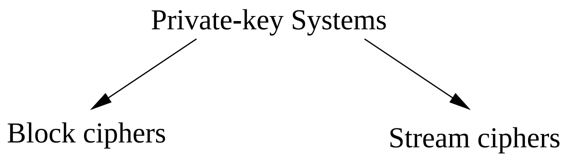


Figure 1.3: Classification of private-key systems

Idea: The message string is divided into blocks (or cells) of equal length that are then encrypted and decrypted.

Input: message string $\bar{X} \to \bar{X} = x_1, x_2, x_3, \ldots, x_n$, where each x_i is one block.

Cipher: $\bar{Y} = y_1, y_2, y_3, \ldots, y_n$; with $y_i = e_k(x_i)$ where the key k is fixed.

1.6.1 Shift Cipher

One of the most simple ciphers where the letters of the alphabet are assigned a number as depicted in Table 1.1.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Table 1.1: Shift cipher table

Definition 1.6.1 Shift Cipher

Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = Z_{26}$. $x \in \mathcal{P}$, $y \in \mathcal{C}$, $k \in \mathcal{K}$.

Encryption: $e_k(x) = x + k \mod 26$.

Decryption: $d_k(y) = y - k \mod 26$.

Remark:

If $k = 3$ the the shift cipher is given a special name — "Caesar Cipher".

Example:

$k = 17$,

plaintext:

$X = x_1, x_2, \ldots, x_6 = ATTACK.$

$X = x_1, x_2, \ldots, x_6 = 0, 19, 19, 0, 2, 10.$

encryption:

$y_1 = x_1 + k \mod 26 = 0 + 17 = 17 \mod 26 = R$

$y_2 = y_3 = 19 + 17 = 36 \equiv 10 \bmod 26 = K$

$y_4 = 17 = R$

$y_5 = 2 + 17 = 19 \bmod 26 = T$

$y_6 = 10 + 17 = 27 \equiv 1 \bmod 26 = B$

ciphertext: $Y \hat{=} y_1, y_2, \ldots, y_6 =$ R K K R T B.

Attacks on Shift Cipher

1. Ciphertext-only: Try all possible keys ($|k| = 26$). This is known as "brute force attack" or "exhaustive search".

   Secure cryptosystems require a sufficiently large key space. Minimum requirement today is $|K| > 2^{80}$, however for long-term security, $|K| \geq 2^{100}$ is recommended.

2. Same cleartext maps to same ciphertext $\Rightarrow$ can also easily be attacked with letter-frequency analysis.

1.6.2 Affine Cipher

This cipher is an extension of the Shift Cipher ($y_i = x_i + k \bmod m$).

Definition 1.6.2 Affine Cipher *Let* $P = C = Z_{26}$.

encryption: $e_k(x) = a \cdot x + b \bmod x$.

key: $k = (a, b)$ *where* $a, b \in Z_{26}$.

decryption: $a \cdot x + b = y \bmod 26$.

$a \cdot x = (y - b) \bmod 26$.

$x = a^{-1} \cdot (y - b) \bmod 26$.

restriction: $gcd(a, 26) = 1$ *in order for the affine cipher to work since* a^{-1} *does not always exist.*

Question: How is a^{-1} obtained?

Answer: $a^{-1} \equiv a^{11} \bmod 26$ (the proof for this is in Chapter 6) or by trial-and-error for the time being.

Chapter 2

Stream Ciphers

Further Reading: [Sim92, Chapter 2]

2.1 Introduction

Remember classification:

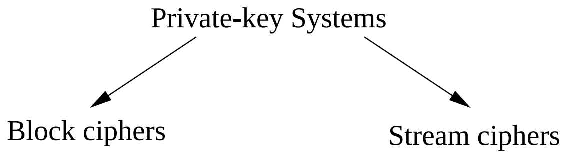


Figure 2.1: Private-key cipher classification

Block Cipher: $\bar{Y} = y_1, y_2, \ldots, y_n = e_k(x_1), e_k(x_2), \ldots, e_k(x_n)$,
e.g. the key does not change with every block

Stream Cipher: $\bar{Y} = y_1, y_2, \ldots, y_n = e_{z_1}(x_1), e_{z_2}(x_2), \ldots, e_{z_n}(x_n)$
with the "keystream" $= z_1, z_2, \ldots, z_n$

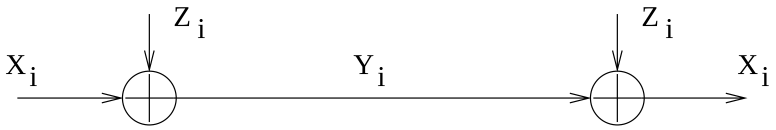


Figure 2.2: Most Popular Encryption/Decryption Function

Most popular en/decryption function: modulo 2 addition

Assume: $x_i, y_i, z_i \in \{0, 1\}$

$$\boxed{\begin{aligned} &y_i = e_{z_i}(x_i) = x_i + z_i \bmod 2 \rightarrow \text{encryption} \\ &x_i = e_{z_i}(y_i) = y_i + z_i \bmod 2 \rightarrow \text{decryption} \end{aligned}}$$

Remarks:

1. Developed by Vernam in 1917 for Baudot Code on teletypewriters.

2. The modulo 2 operation is equivalent to a 2-input XOR operation.

   Why are encryption and decryption identical operations? Truth table of modulo 2 addition:

a	b	$c = a + b \bmod 2$
0	0	$0 + 0 = 0 \bmod 2$
0	1	$0 + 1 = 1 \bmod 2$
1	0	$1 + 0 = 1 \bmod 2$
1	1	$1 + 1 = 0 \bmod 2$

.

   $\Rightarrow$ modulo 2 addition yields the same truth table as the XOR operation.

3. Encryption and decryption are the same operation, namely modulo 2 addition (or XOR).

   Why? We show that decryption of ciphertext bit y_i yields the corresponding plaintext bit.

Decryption: $y_i + z_i = \underbrace{(x_i + z_i)}_{\text{encryption}} + z_i = x_i + (z_i + z_i) \equiv x_i \bmod 2.$

Note that $z_i + z_i \equiv 0 \bmod 2$ for $z_i = 0$ and for $z_i = 1$.

Example: Encryption of the letter 'A' by Alice.

'A' is given in ASCII code as $65_{10} = 1000001_2$.

Let's assume that the first key stream bits are $\rightarrow z_1, \ldots, z_7 = 0101101$

Encryption by Alice:	plaintext x_i:	1000001	= 'A'	(ASCII symbol)
	key stream z_i:	0101101		
	ciphertext y_i:	1101100	= 'l'	(ASCII symbol)
Decryption by Bob:	ciphertext y_i:	1101100	= 'l'	(ASCII symbol)
	key stream z_i:	0101101		
	plaintext x_i:	1000001	= 'A'	(ASCII symbol)

2.2 One-Time Pad and Pseudo-Random Generators

Definition 2.2.1 *Unconditional Security*

A cryptosystem is unconditionally *secure if it cannot be broken even with infinite computational resources.*

Definition 2.2.2 *One-time Pad (OTP)*

A cryptosystem developed by Mauborgne based on Vernam's stream cipher consisting of:

$|\mathcal{P}| = |\mathcal{C}| = |\mathcal{K}|$,

with $x_i, y_i, k_i \in \{0, 1\}$.

encrypt $\rightarrow e_{k_i}(x_i) = x_i + k_i \bmod 2$.

decrypt $\rightarrow d_{k_i}(y_i) = y_i + k_i \bmod 2$.

Theorem 2.2.1 *The OTP is unconditionally secure if keys are only used once.*

Remarks:

1. OTP is the only provable secure system:

$$\begin{aligned} y_0 &= x_0 + K_0 \bmod 2 \\ y_1 &= x_1 + K_1 \bmod 2 \\ &\vdots \end{aligned}$$

   each equality is a linear equation with 2 unknowns.

   $\Rightarrow$ for every y_i, $x_i = 0$ and $x_i = 1$ are equally likely.

   $\Rightarrow$ holds only if $K_0, K_1, \ldots$ are not related to each other, i.e., K_i must be generated trully randomly.

2. OTP are impractical for most applications.

Question: Can we "emulate" a OTP by using a short key?

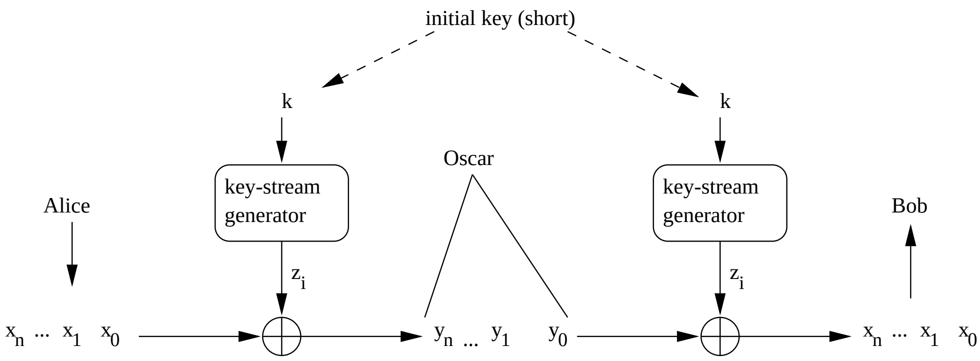


Figure 2.3: Stream cipher model

Classification by key-stream generator:

a) "synchronous stream cipher"

$z_i = f(k) \rightarrow$ pseudo-random generator (PRG).

b) "asynchronous stream cipher"

$z_i = f(k, y_{i-1}, y_{i-2}, \ldots, y_{i-N}) \rightarrow$ feedback of cipher.

c) The key issue is that Bob has to 'match' the exact z_i to get the correct message. In order to do this, both key-stream generators have to be synchronized.

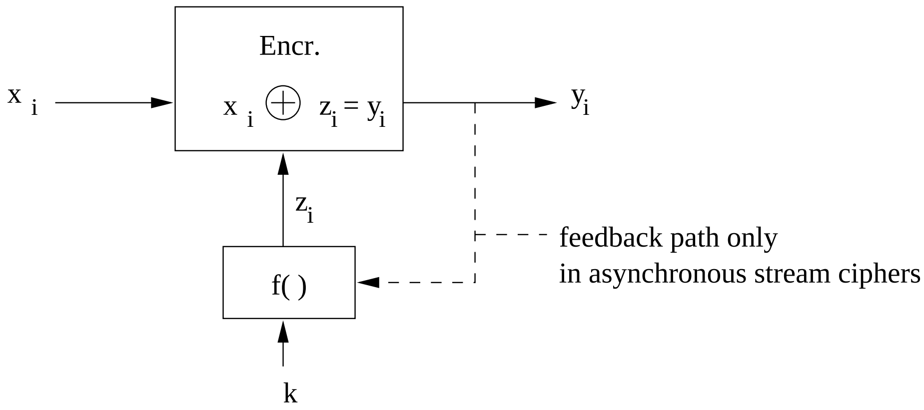


Figure 2.4: Asynchronous stream cipher

It is important to note that key stream generators must not only possess good statistical properties, which is true for other pseudo-random generatores as well, but they must also be *cryptographically secure*:

> **Definition 2.2.3** *Cryptographically secure pseudo-random generators*
> *A pseudo random generator (key stream generator) is* cryptographically secure *if it is unpredictable. That is, given the first n output bits of the generator, it is computatinally infeasible to compute the bits $n+1, n+2, \ldots$*

2.3 Synchronous Stream Ciphers

The keystream $z_1, z_2, \ldots$ is a pseudo-random sequence which depends only on the key.

2.3.1 Linear Feedback Shift Registers (LFSR)

An LFSR consists of m storage elements (flip-flops) and a feedback network. The feedback network computes the input for the "last" flip-flop as XOR-sum of certain flip-flops in the shift register.

Example: We consider an LFSR of degree $m = 3$ with flip-flops K_2, K_1, K_0, and a feedback path as shown below.

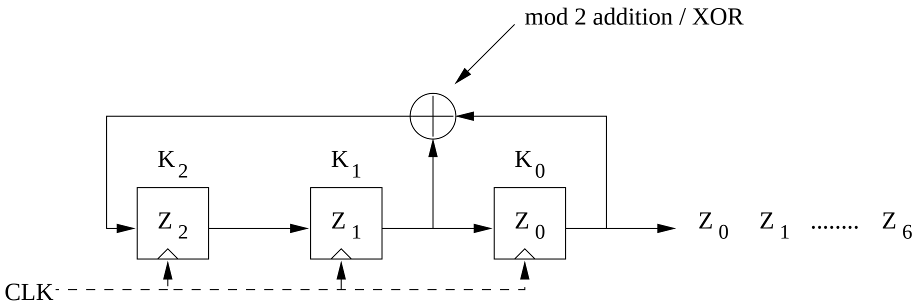


Figure 2.5: Linear feedback shift register

K_2	K_1	K_0
1	0	0
0	1	0
1	0	1
1	1	0
1	1	1
0	1	1
0	0	1
1	0	0

Mathematical description for keystream bits z_i with z_0, z_1, z_2 as initial settings:

$$z_3 = z_1 + z_0 \bmod 2$$

$$z_4 = z_2 + z_1 \bmod 2$$

$$z_5 = z_3 + z_2 \bmod 2$$

$$\vdots$$

general case: $z_{i+3} = z_{i+1} + z_i \bmod 2;\ i = 0, 1, 2, \ldots$

Expression for the LFSR:

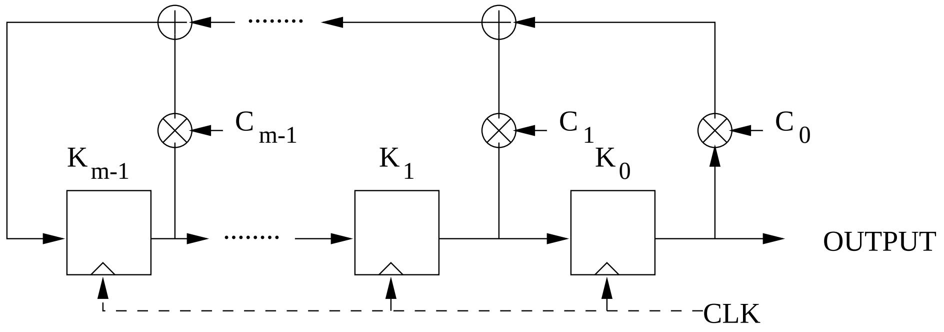


Figure 2.6: LFSR with feedback coefficients

$C_0, C_1, \ldots, C_{m-1}$ are the *feedback coefficients*. $C_i = 0$ denotes an open switch (no connection), $C_i = 1$ denotes a closed switch (connection).

$$\boxed{z_{i+m} = \sum_{j=0}^{m-1} C_j \cdot z_{i+j} \bmod 2;\ C_j \in \{0,1\};\ i = 0, 1, 2, \ldots}$$

The entire key consists of:

$$k = \{(C_0, C_1, \ldots, C_{m-1}), (z_0, z_1, \ldots, z_{m-1}), m\}$$

Example:

$$k = \{(C_0 = 1, C_1 = 1, C_2 = 0), (z_0 = 0, z_1 = 0, z_2 = 1), 3\}$$

Theorem 2.3.1 *The maximum sequence length generated by the LFSR is $2^m - 1$.*

Proof:

There are only 2^m different states $(k_0, \ldots, k_m)$ possible. Since only the current state is known to the LFSR, after 2^m clock cycles a repetition must occur. The all-zero state must be excluded since it repeats itself immediately.

Remarks:

1.) Only certain configurations $(C_0, \ldots, C_{m-1})$ yield maximum length LFSRs.

For example:

if $m = 4$ then $(C_0 = 1, C_1 = 1, C_2 = 0, C_3 = 0)$ has length of $2^m - 1 = 15$

but $(C_0 = 1, C_1 = 1, C_2 = 1, C_3 = 1)$ has length of 5

2.) LFSRs are sometimes specified by polynomials.

such that the $P(x) = x^m + C_{m-1}x^{m-1} + \ldots + C_1 x + C_0$.

Maximum length LFSRs have *"primitive polynomials"*.

These polynomials can be easily obtained from literature (Table 16.2 in [Sch93]).

For example:

$(C_0 = 1, C_1 = 1, C_2 = 0, C_3 = 0) \Longleftrightarrow P(x) = 1 + x + x^4$

2.3.2 Clock Controlled Shift Registers

Example: *Alternating stop-and-go generator.*

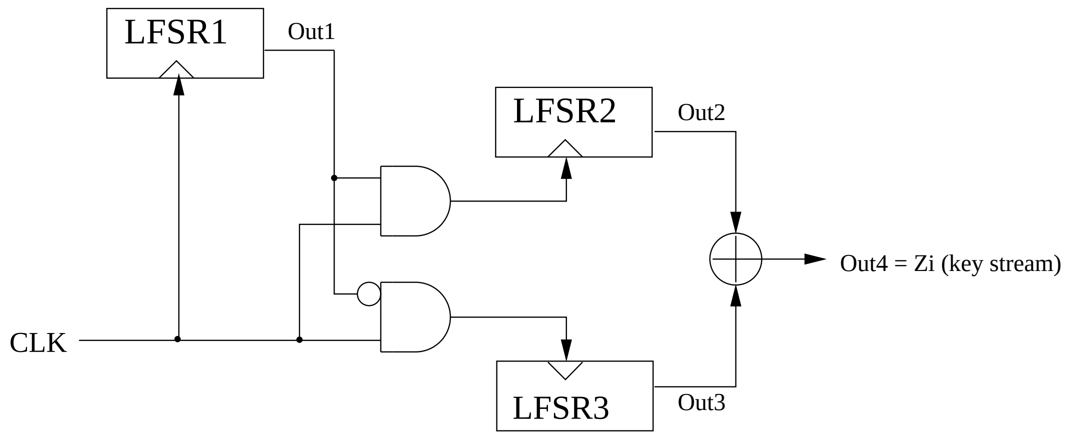


Figure 2.7: Stop-and-go generator example

Basic operation:

When $Out_1 = 1$ then LFSR2 is clocked otherwise LFSR3 is clocked.

Out_4 serves as the keystream and is a bitwise XOR of the results from LFSR2 and LFSR3.

Security of the generator:

- All three LFSRs should have maximum length configuration.
- If the sequence lengths of all LFSRs are relatively prime to each other, then the sequence length of the generator is the product of all three sequence lengths, i.e., $L = L_1 \cdot L_2 \cdot L_3$.
- A secure generator should have LFSRs of roughly equal lengths and the length should be at least 128: $m_1 \approx m_2 \approx m_3 \approx 128$.

2.4 Attacks

2.4.1 Known Plaintext Attack Against LFSRs

Assumption:

For a known plaintext attack, we have to assume that m is known.

Idea:

This attack is based on the knowledge of some plaintext and its corresponding ciphertext.

i) Known plaintext $\rightarrow x_0, x_1, \ldots, x_{2m-1}$.

ii) Observed ciphertext $\rightarrow y_0, y_1, \ldots, y_{2m-1}$.

iii) Construct keystream bits $\rightarrow z_i = x_i + y_i \bmod 2; i = 0, 1, \ldots, 2m-1$.

Goal:

To find the feedback coefficients C_i.

Using the LFSR equation to find the C_i coefficients:

$$z_{i+m} = \sum_{j=0}^{m-1} C_j \cdot z_{i+j} \bmod 2; C_j \in \{0, 1\}$$

We can rewrite this in a matrix form as follows:

$$\begin{array}{llcll} i=0 & z_m & = & C_0 z_0 + C_1 z_1 + \ldots + C_{m-1} z_{m-1} & \bmod 2. \\ i=1 & z_{m+1} & = & C_0 z_1 + C_1 z_2 + \ldots + C_{m-1} z_m & \bmod 2. \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ i=m-1 & z_{2m-1} & = & C_0 z_{m-1} + C_1 z_m + \ldots + C_{m-1} z_{2m-2} & \bmod 2. \end{array} \tag{2.1}$$

Note:

We now have m linear equations in m unknowns $C_0, C_1, \ldots, C_{m-1}$. The C_i coefficients are constant making it possible to solve for them when we have $2m$ plaintext-ciphertext pairs.

Rewriting Equation (2.1) in matrix form, we get:

$$\begin{bmatrix} z_0 & \dots & z_{m-1} \\ \vdots & & \vdots \\ z_{m-1} & \dots & z_{2m-2} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} z_m \\ \vdots \\ z_{2m-1} \end{bmatrix} \bmod 2 \tag{2.2}$$

Solving the matrix in (2.2) for the C_i coefficients we get:

$$\begin{bmatrix} c_0 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} z_0 & \dots & z_{m-1} \\ \vdots & & \vdots \\ z_{m-1} & \dots & z_{2m-2} \end{bmatrix}^{-1} \cdot \begin{bmatrix} z_m \\ \vdots \\ z_{2m-1} \end{bmatrix} \bmod 2 \tag{2.3}$$

Summary:

> By observing $2m$ output bits of an LFSR of degree m and matching them to the known plaintext bits, the C_i coefficients can exactly be constructed by solving a system of linear equations of degree m.

$\Rightarrow$ **LFSRs by themselves are extremely un-secure!** However, combinations of them such as the Alternating stop-and-go generator **can** be secure.

Chapter 3

Some Results From Information Theory

3.1 Levels of Security

Definition 3.1.1 *Unconditional Security*
A cryptosystem is unconditionally *secure if it cannot be broken even with infinite computational resources.*

Theorem 3.1.1 *The OTP is unconditionally secure if keys are only used once.*

3.2 Computational Security

For **all** known practical cryptosystems we have:

Definition 3.2.1 *Computational Security*
*A system is "computational secure" if the **best possible algorithm** for breaking it requires N operations, where N is very large and known.*

Unfortunately, **all** known practical systems are only computational secure for **known algorithms**.

Definition 3.2.2 *Relative Security*   *A system is "relative secure" if its security relies on a well studied, very hard problem.*

Example:

> A system S is secure as long as factoring of large integers is hard (this is believed for RSA).

3.3 Cryptography and Coding

There are three basic forms of coding in modern communication systems: source coding, channel coding, and encryption. From an information theoretical and practical point of view, the three forms of coding should be applied as follows:

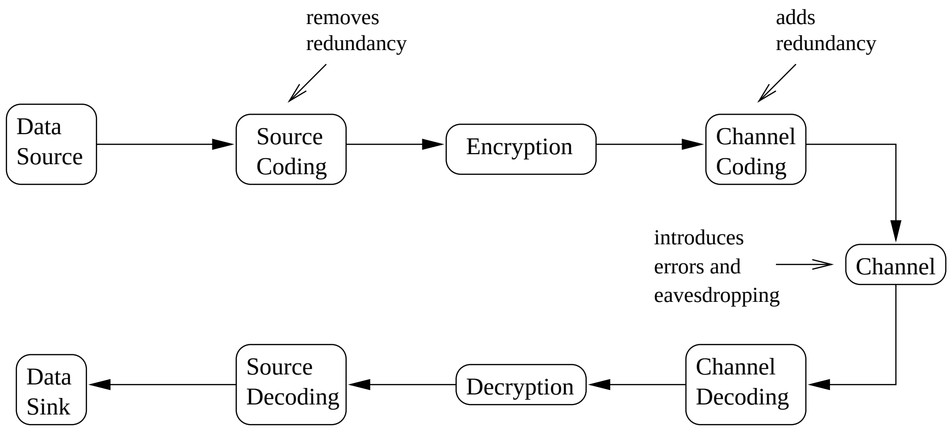


Figure 3.1: Communication coding system model

3.4 Confusion and Diffusion

According to Shannon, there are two basic approaches to encryption.

1. **Confusion** — encryption operation where the relationship between cleartext and ciphertext is obscured. Some examples are:

   (a) Shift cipher — main operation is substitution.

   (b) German Enigma (broken by Turing) — main operation is *smart* substitution.

2. **Diffusion** — encryption by spreading out the influence of one cleartext letter over many ciphertext letters. An example is:

   (a) permutations — changing the positioning of the cleartext.

Remarks:

1. Today $\rightarrow$ changing of one bit of cleartext should result on average in the change of half the output bits.

   $x_1 = 001010 \rightarrow$ *encr.* $\rightarrow y_1 = 101110.$

   $x_2 = 000010 \rightarrow$ *encr.* $\rightarrow y_2 = 001011.$

2. Combining confusion with diffusion is a common practice for obtaining a secure scheme. Data Encryption Standard (DES) is a good example of that.

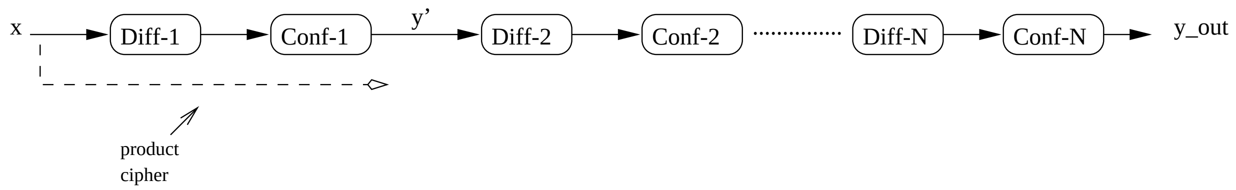


Figure 3.2: Example of combining confusion with diffusion

Chapter 4

Data Encryption Standard (DES)

General Notes:

- DES is by far the most popular private-key algorithm.
- It was published in 1975 and standardized in 1977.
- Expired in 1998.

4.1 Encryption

System Parameters:

→ block cipher.

→ 64 input/output bits.

→ 56 bits of key.

Principle: 16 rounds of encryption.

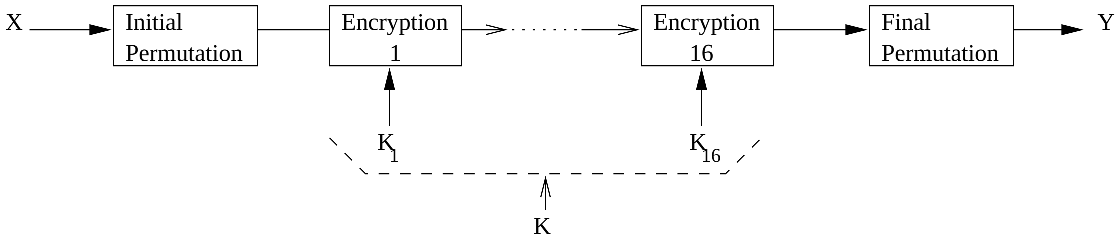


Figure 4.1: General Model of DES

4.1.1 Overview

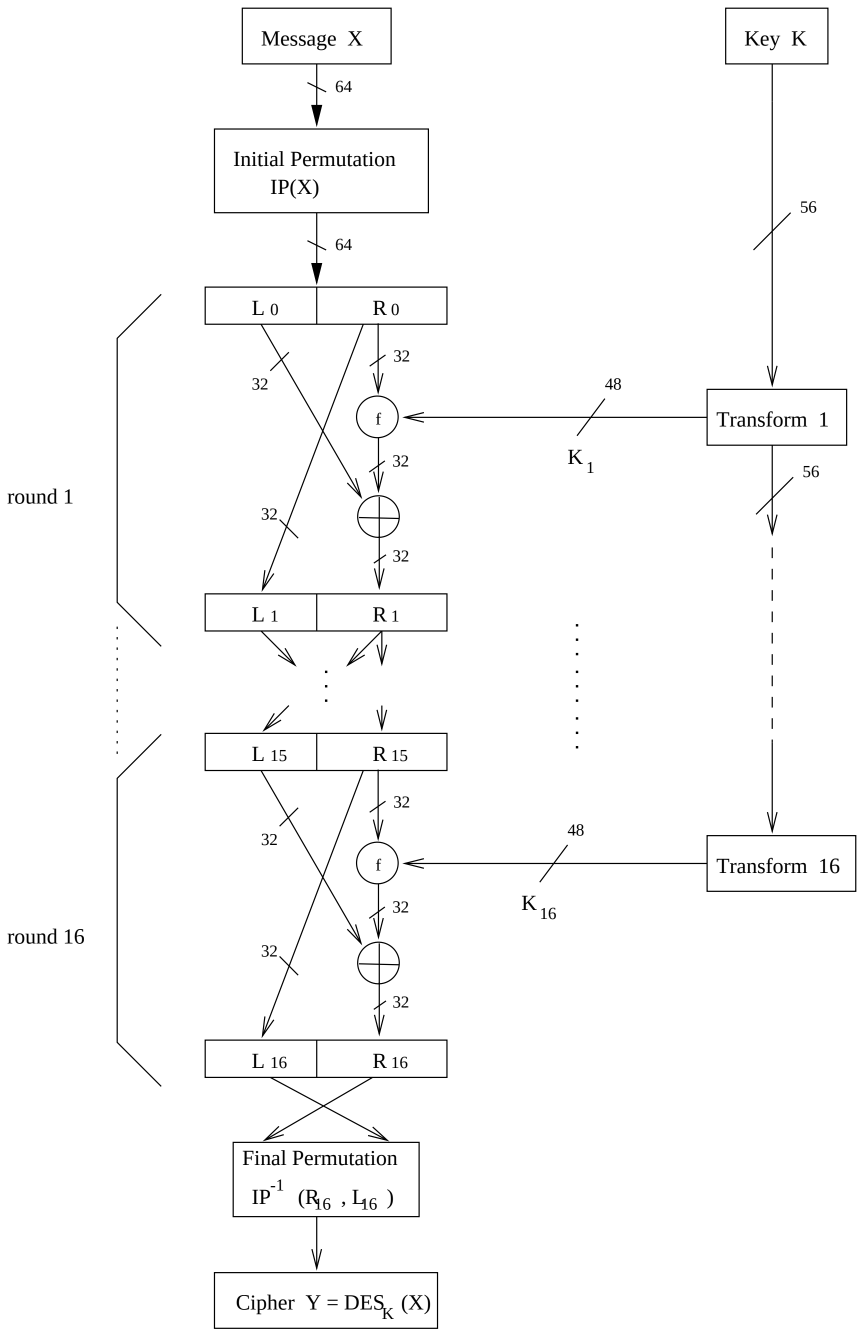


Figure 4.2: The Feistel Network

4.1.2 Permutations

a) Initial Permutation IP.

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

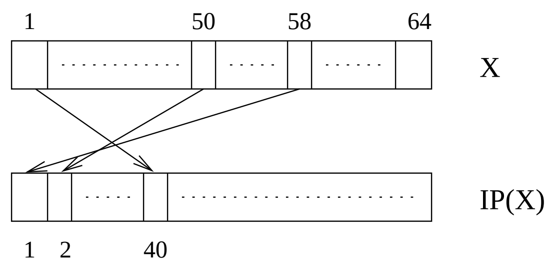


Figure 4.3: Initial permutation

b) Inverse Initial Permutation IP^{-1} (final permutation).

Note:

$IP^{-1}(IP(X)) = X.$

4.1.3 Core Iteration / f-Function

General Description:

$L_i = R_{i-1}.$

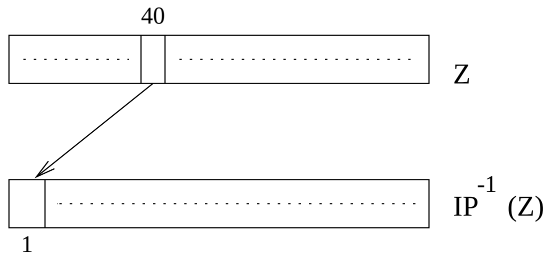


Figure 4.4: Final permutation

$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$.

The core iteration is the f-function that takes the right half of the output of the previous round and the key as input.

E bit table					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

S-boxes:

Contain look-up tables (LUTs) with 64 numbers ranging from $0 \ldots 15$.

Input: Six bit code selecting one number.

Output: Four bit binary representation of one number out of 64.

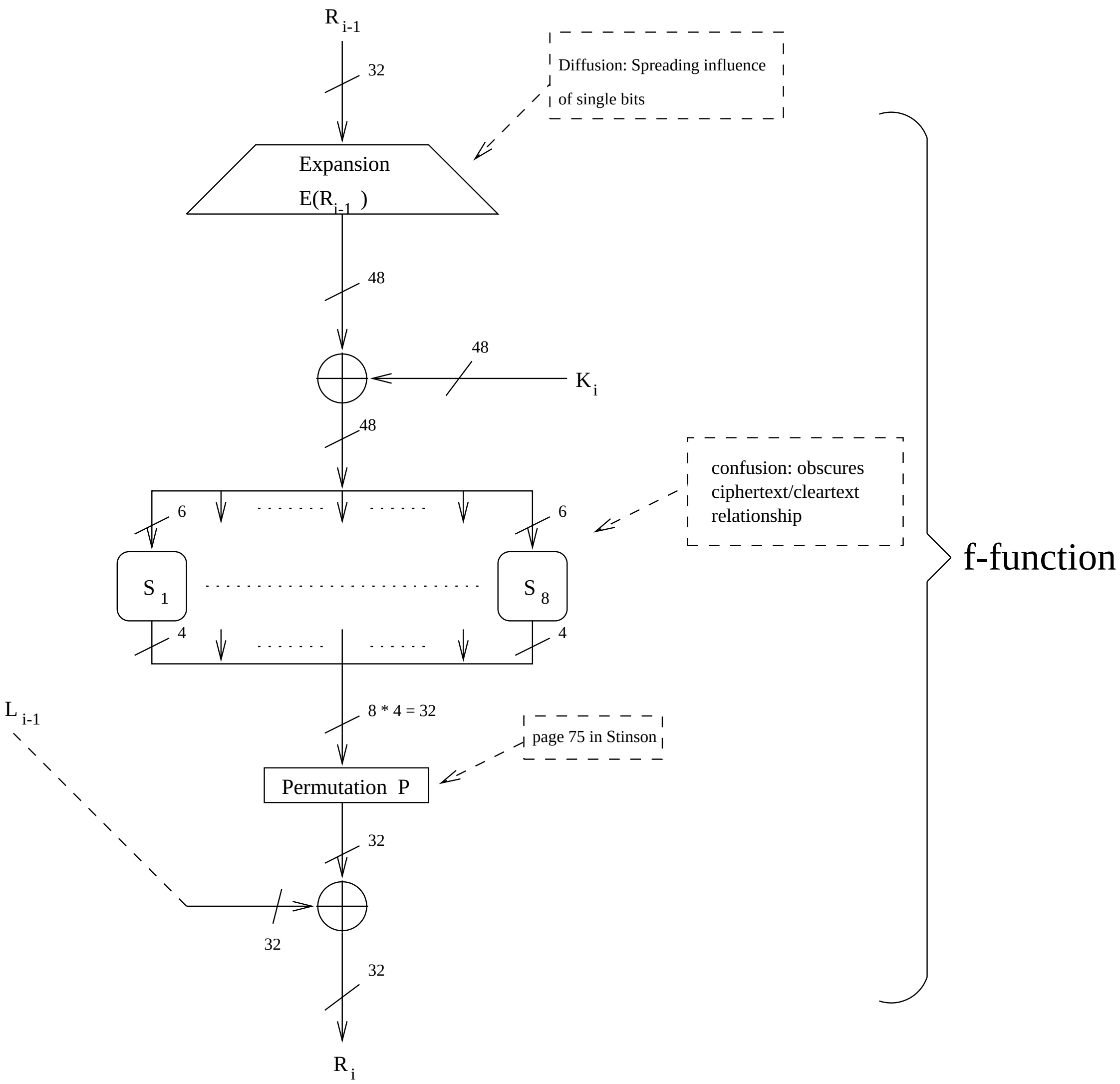


Figure 4.5: Core function of DES

Example:

S1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Box 1

Input: Six bit vector with MSB and LSB selecting the row and four inner bits selecting column.

$b = (100101)$.

$\rightarrow$ row $= (11)_2 = 3$ (forth row).

$\rightarrow$ column $= (0010)_2 = 2$ (third column).

$S_1(37 = 100101_2) = 8 = 1000_2$.

Remark:

S-boxes are the most crucial elements of DES because they introduce a **non-linear** function to the algorithm, i.e., $S(a)$ XOR $S(b) \neq S(a$ XOR $b)$.

4.1.4 Key Schedule

Note:

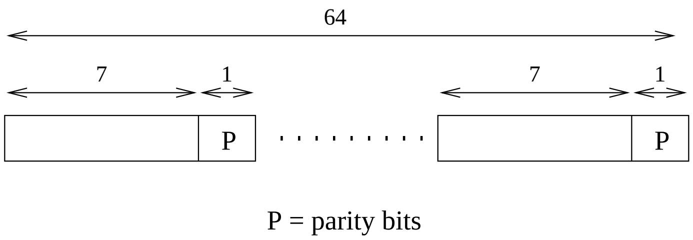


Figure 4.6: 64 bit DES block

In practice the DES key is artificially enlarged with odd parity bits. These bits are "stripped" in PC-1.

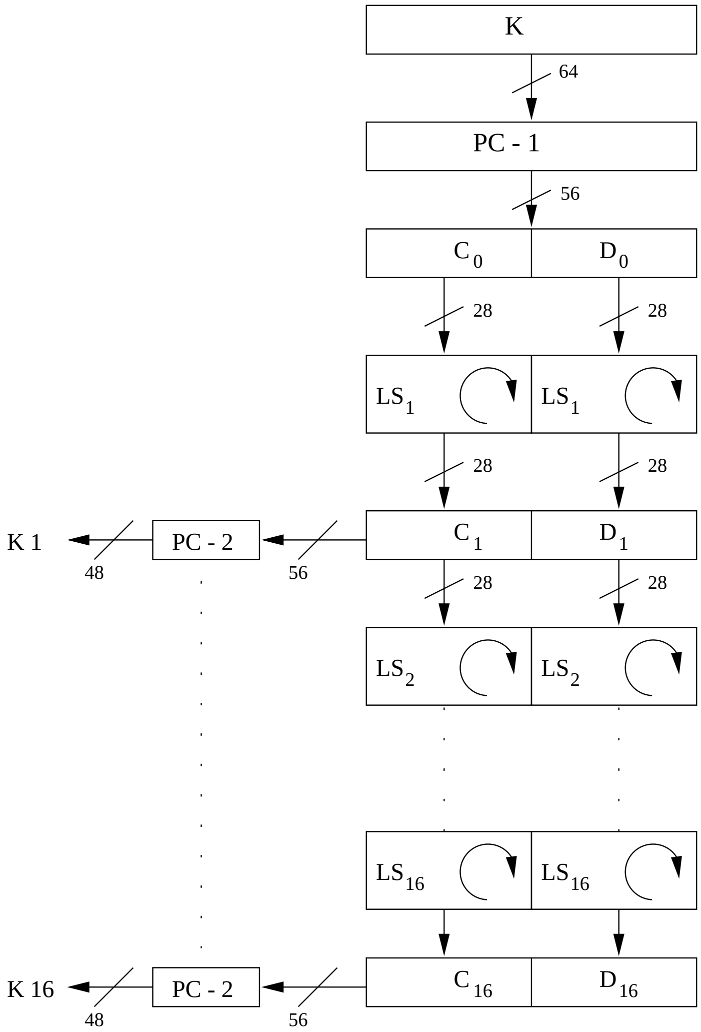


Figure 4.7: DES key scheduler

The cyclic Left-Shift (LS) blocks have two modes of operation:

a) for LS_i where $i = 1, 2, 9, 16$, the block is shifted once.

b) for LS_i where $i \neq 1, 2, 9, 16$, the block is shifted twice.

Remark:

The total number of cyclic Left-Shifts is $4 \cdot 1 + 12 \cdot 2 = 28$. As a results of this $C_0 = C_{16}$ and $D_0 = D_{16}$.

4.2 Decryption

One advantage of DES is that decryption is essentially the same as encryption. Only the key schedule is reversed. This is due to the fact that DES is based on a Feistel network.

Question: Why does decryption work essentially the same as encryption?

a) Find what happens in the initial stage of decryption!

$(L_0^d, R_0^d) = IP(Y) = IP(IP^{-1}(R_{16}, L_{16})) = (R_{16}, L_{16})$.

$(L_0^d, R_0^d) = IP(Y) = (R_{16}, L_{16})$.

$L_0^d = R_{16}$.

$R_0^d = L_{16} = R_{15}$.

b) Find what happens in the iterations!

What are (L_1^d, R_1^d) ?

$L_1^d = R_0^d = L_{16} = R_{15}$.

substitute into the above equation to get:

$R_1^d = L_0^d \oplus f(R_0^d, k_{16}) = R_{16} \oplus f(L_{16}, k_{16})$.

$R_1^d = [L_{15} \oplus f(R_{15}, k_{16})] \oplus f(R_{15}, k_{16})$.

$R_1^d = L_{15} \oplus [f(R_{15}, k_{16}) \oplus f(R_{15}, k_{16})] = L_{15}$.

in general: $L_i^d = R_{16-i}$ and $R_i^d = L_{16-i}$;

such that: $L_{16}^d = R_{16-16} = R_0$ and $R_{16}^d = R_0$.

c) Find what happens in the final stage!

$IP^{-1}(R_{16}^d, L_{16}^d) = IP^{-1}(L_0, R_0) \doteq IP^{-1}(IP(X)) = X$ *q.e.d.*

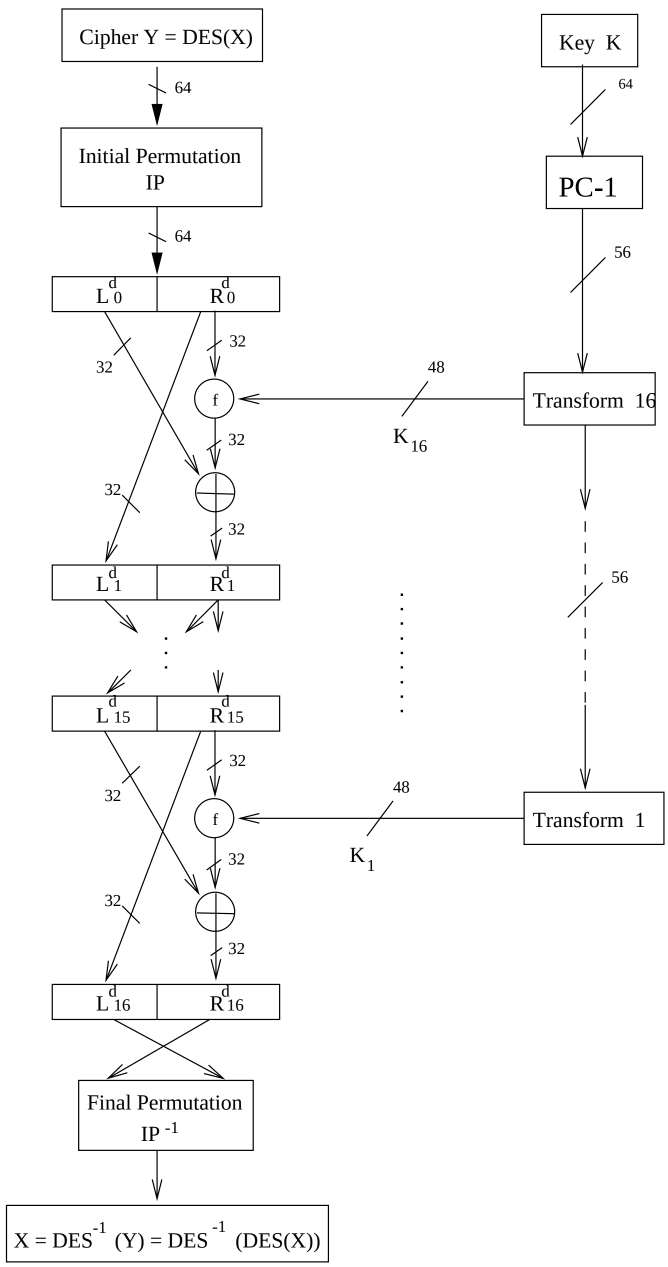


Figure 4.8: Decryption of DES

<u>Reversed Key Schedule:</u>

Question: Given K, how can we easily generate k_{16}?

$k_{16} = PC2(C_{16}, D_{16}) = PC2(C_0, D_0) = PC2(PC1(k))$.

$k_{15} = PC2(C_{15}, D_{15}) = PC2(RS_1(C_{16}), RS_1(D_{16})) = PC2(RS_1(C_0), RS_1(D_0))$.

4.3 Implementation

Note:

One design criteria for DES was fast hardware implementation.

4.3.1 Hardware

Since permutations and simple table look-ups are fast in hardware, DES can be implemented very efficiently [AM97, page 362].

Fastest Implementation:

$\Rightarrow$ 9 Gbit/s as 0.6 μm technology ASIC [WPR$^+$99] with 16 stage pipeline.

4.3.2 Software

Record: 130 Mbits/s by Biham [Bih97].

Typically: a few 10 Mbit/s.

4.4 Attacks

There have been two major points of criticism about DES from the beginning:

i) key size is too small,

ii) the S-boxes contained secret design criteria.

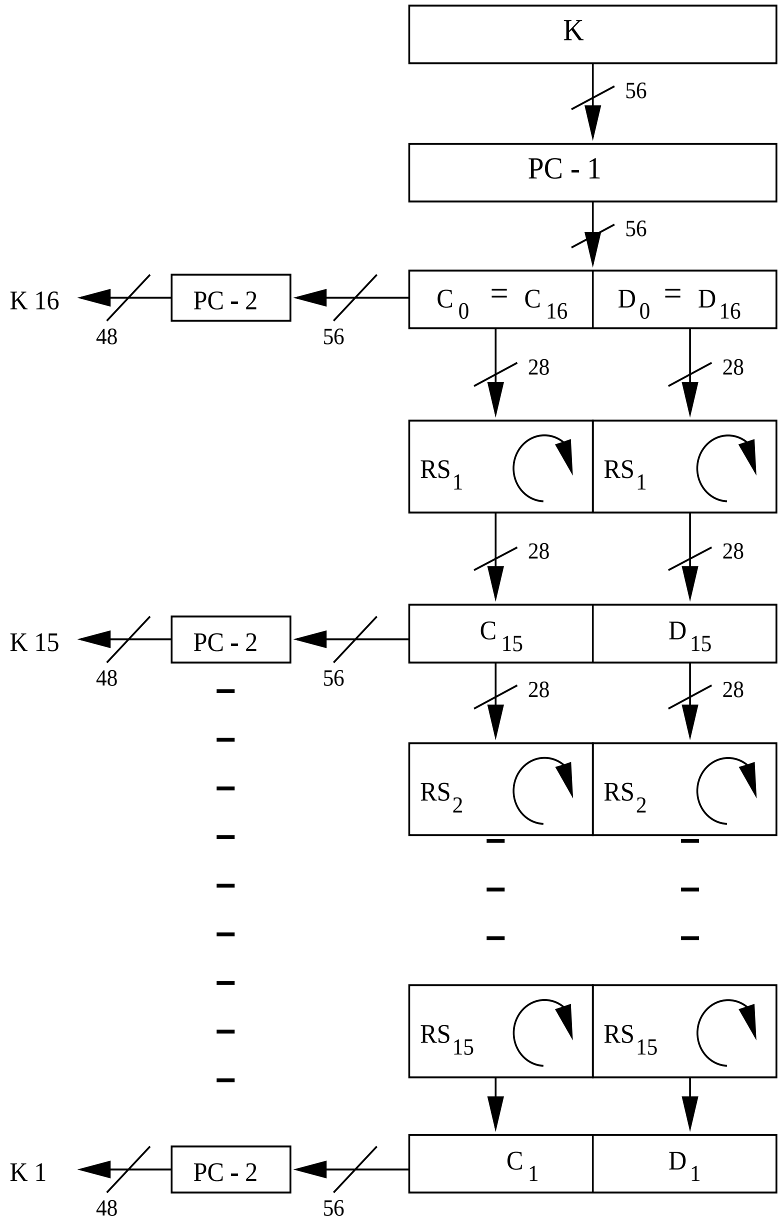


Figure 4.9: Reversed key scheduler for decryption of DES

4.4.1 Exhaustive Key Search

Known Plaintext Attack:

known: X and Y.

unknown: K, such that $Y = DES_k(X)$.

idea: test all 2^{56} possible keys $\rightarrow DES_{k_i}(X) \stackrel{?}{=} Y;\ i = 0, 1, \ldots, 2^{56} - 1$.

4.4.2 Differential Cryptanalysis

Proposed by Biham/Shamir in 1990.

Principle:

To consider differences in plain and ciphertext pairs and deduce the likelihood of certain <u>keys</u>.

16-round DES requirements:

With chosen plaintext, 2^{47} (X,Y) pairs are needed.

With known plaintext, 2^{55} (X,Y) pairs are needed.

2^{37} arithmetic operations are needed.

Since each (X,Y) pair is 128 bits long, large storage is needed which makes this attack highly impractical!

Remark: The DES S-boxes are optimized against differential cryptanalysis.

4.4.3 Linear Cryptanalysis

Proposed by Matsui in 1993 and presented at CRYPTO'94.

Principal:

To consider differences in plain and ciphertext pairs and deduce the likelihood of certain key <u>bits</u>.

The actual attack was implemented:

$\rightarrow$ with 2^{43} known plaintexts, the key was recovered in 50 days.

$\rightarrow$ using 12 HP RISC workstations running at 99MHz.

Remark: The S-box design of DES is not optimized for this attack.

Date	Proposed/implemented attack
1977	Diffie & Hellman, estimate cost of key search machine (underestimate)
1990	Biham & Shamir propose differential cryptoanalysis (2^{47} chosen ciphertexts)
1993	Mike Wiener proposes detailed hardware design for key search machine: average search time of 36 h @ $100,000
1993	Matsui proposes linear cryptoanalysis (2^{43} chosen ciphertexts)
Jun. 1997	DES Challenge I broken, distributed effort took 4.5 months
Feb. 1998	DES Challenge II–1 broken, distributed effort took 39 days
Jul. 1998	DES Challenge II–2 broken, key-search machine built by the Electronic Frontier Foundation (EFF), 1800 ASICs, each with 24 search units, $250K, 15 days average (actual time 56 hours)
Jan. 1999	DES Challenge III broken, distributed effort combined with EFF's key-search machine, it took 22 hours and 15 minutes.

Table 4.1: History of full-round DES attacks

4.5 DES Alternatives

There exists a wealth of other block ciphers. A small collection of as of yet unbroken ciphers is:

Algorithm	*Year*	*Inventor*	*X/Y bits*	*Key*	*Core Operation*
AES	2000+	?	128	128/192/256	?
Triple DES			64	112	S-box
IDEA	90/92	Lai/Massey	64	128	modulo arithmetic
Cast	93	Adams/Tavares	64	64	variable S-boxes
Safer	94	Massey	64	64/128	modulo arithmetic

For further reading, consult Chapters 13 and 14 in [Sch93].

Chapter 5

Rijndael – The Advanced Encryption Standard

5.1 History

5.1.1 Basic Facts about AES

- Successor to DES.
- The AES selection process was administered by NIST.
- Unlike DES, the AES selection was an open (i.e., public) process.
- Likely to be the dominant secret-key algorithm in the next decade.
- Main AES requirements by NIST:
  - Block cipher with 128 I/O bits
  - Three key lengths must be supported: 128/192/256 bits
  - Security relative to other submitted algorithms
  - Efficient software and hardware implementations

- See `http://www.nist.gov/aes` for further information on AES

5.1.2 Chronology of the AES Process

- Development announced on January 2, 1997 by the National Institute of Standards and Technology (NIST).
- 15 candidate algorithms accepted on August 20th, 1998.
- 5 finalists announced on August 9th, 1999
  - Mars, IBM Corporation.
  - RC6, RSA Laboratories.
  - Rijndael, J. Daemen & V. Rijmen.
  - Serpent, Eli Biham et al.
  - Twofish, B. Schneier et al.
- Monday October 2nd, 2000, NIST chooses Rijndael as the AES.

A lot of work went into software and hardware performance analysis of the AES candidate algorithms. Here are representative numbers:

Algorithm	Pentium-Pro @ 200 MHz (Mbit/sec) [WWGP00]	FPGA Hardware (Gbit/sec) [EYCP00]
MARS	69	–
RC6	105	2.4
Rijndael	71	1.9
Serpent	27	4.9
Twofish	95	1.6

Table 5.1: Speeds of the AES Finalists in Hardware and Software

5.2 Rijndael Overview

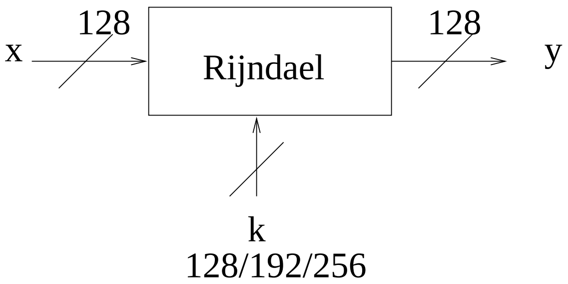


Figure 5.1: AES Block and Key Sizes

- Both blocksize and keylength of Rijndael are variable. Sizes shown in Figure 5.2 are the ones required by the AES Standard. The number of rounds (or iterations) is a function of the key length:

Key lengths (bits)	n_r = # rounds
128	10
192	12
256	14

Table 5.2: Key lenghts and number of rounds for Rijndael

- However, Rijndael also allows *blocksizes* of 192 and 256 bits. For those blocksizes the number of rounds must be increased.

Important: Rijndael does *not* have a Feistel structure. Feistel networks do not encrypt an entire block per iteration (e.g., in DES, $64/2 = 32$ bits are encrypted in one iteration). Rijndael encrypts all 128 bits in one iteration. As a consequence, Rijndael has a comparably small number of rounds.

Rijndael uses three different types of layers. Each layer operates on all 128 bits of a block:

1. *Key Addition Layer:* XORing of subkey.
2. *Byte Substitution Layer:* 8-by-8 SBox substitution.
3. *Diffusion Layer:* provides difussion over all 128 (or 192 or 256) block bits. It is split in two sub-layers:
   (a) ShiftRow Layer.
   (b) MixColumn Layer.

Remark: The ByteSubstitution Layer introduces confusion with a non-linear operation. The ShiftRow and MixColumn stages form a linear Diffusion Layer.

5.3 Some Mathematics: A Very Brief Introduction to Galois Fields

"Galois fields" are used to perform substitution and diffusion in Rijndael.

Question: What are Galois fields?

Galois fields are *fields* with a finite number of elements. Roughly speaking, a field is a structure in which we ca add, subtract, multiply, and compute inverses. More exactly a field is a ring in which all elements except 0 are invertible.

Fact 5.3.1 *Let p be a prime. $GF(p)$ is a "prime field," i.e., a Galois field with a prime number of elements. All arithmetic in $GF(p)$ is done modulo p.*

Example: $GF(3) = \{0, 1, 2\}$

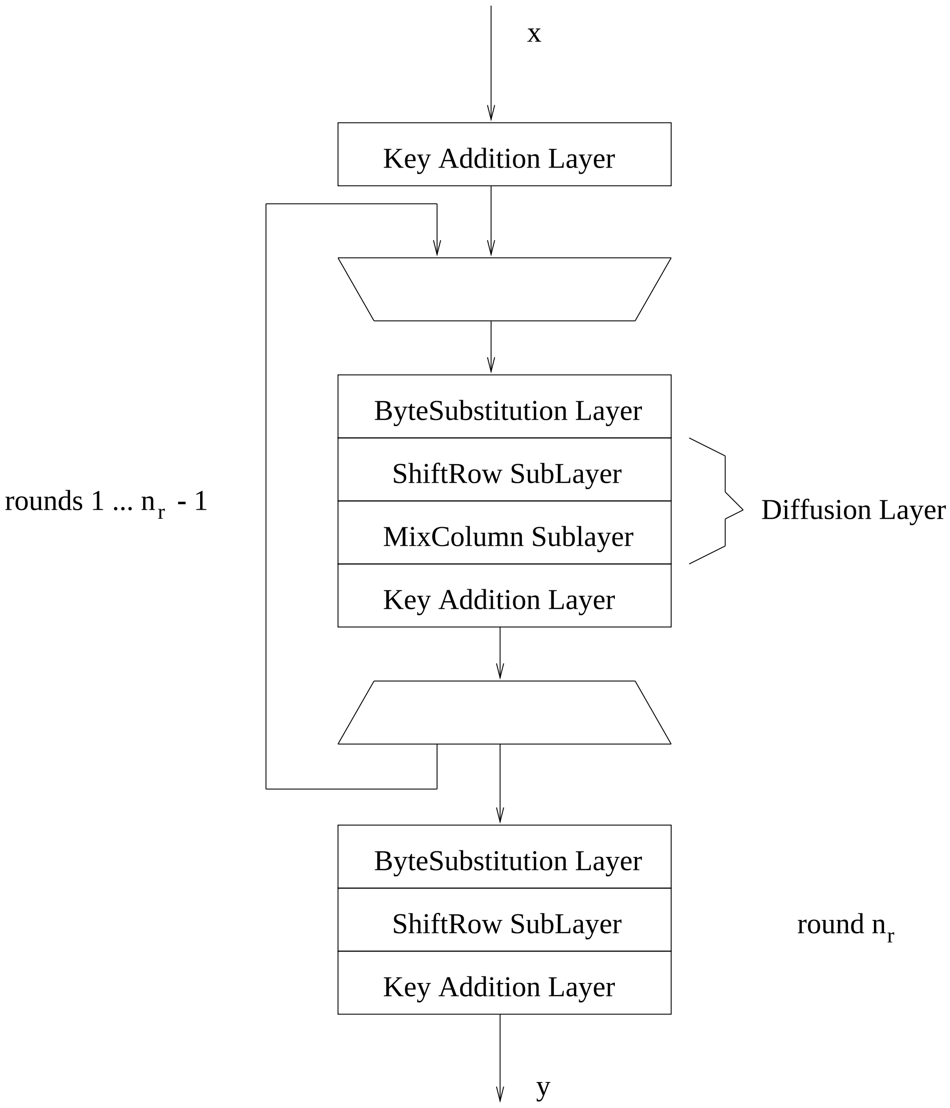


Figure 5.2: Rijndael encryption block diagram

addition

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

additive inverse

$-0 = 0$

$-1 = 2$

$-2 = 1$

multiplication

$\times$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

multiplicative inverse

0^{-1} does not exist

$1^{-1} = 1$

$2^{-1} = 2$, since $2 \cdot 2 \equiv 1 \bmod 3$

Theorem 5.3.1 *For every power p^m, p a prime and m a positive integer, there exists a finite field with p^m elements, denoted by $GF(p^m)$.*

Examples:

- $GF(5)$ is a finite field.
- $GF(256) = GF(2^8)$ is a finite field.
- $GF(12) = GF(3 \cdot 2^2)$ is **NOT** a finite field (in fact, the notation is already incorrect and you should pretend you never saw it).

Question: How to build "extension fields" $GF(p^m)$, $m > 1$?

Note: See also [Sti95, Section 5.2.1]

1. **Represent** elements as polynomials with m coefficients. Each coefficient is an element of $GF(p)$.

   Example: $A \in GF(2^8)$

   $A \rightarrow A(x) = a_7 x^7 + \cdots + a_1 x + a_0, \quad a_i \in GF(2) = \{0, 1\}$

2. **Addition and subtraction in** $GF(p^m)$

   $C(x) = A(x) + B(x) = \sum_{i=0}^{i=m-1} c_i x^i, \quad c_i = a_i + b_i \bmod p$

   Example: $A, B \in GF(2^8)$

$$
\begin{array}{lllllll}
A(x) & = x^7+ & x^6+ & x^4+ & & 1 \\
B(x) & = & & x^4+ & x^2+ & 1 \\
\hline
C(x) & = x^7+ & x^6+ & & x^2 &
\end{array}
$$

3. **Multiplication in** $GF(p^m)$: multiply the two polynomials using polynomial multiplication rule, with coefficient arithmetic done in $GF(p)$. The resulting polynomial will have degree $2m-2$.

$$
\begin{aligned}
A(x) \cdot B(x) &= (a_{m-1}x^{m-1} + \cdots + a_0) \cdot (b_{m-1}x^{m-1} + \cdots + b_0) \\
C'(x) &= c'_{2m-2}x^{2m-2} + \cdots + c'_0
\end{aligned}
$$

where:

$$
\begin{aligned}
c'_0 &= a_0 b_0 \bmod p \\
c'_1 &= a_0 b_1 + a_1 b_0 \bmod p \\
&\vdots \\
c'_{2m-2} &= a_{m-1} b_{m-1} \bmod p
\end{aligned}
$$

Question: How to reduce $C'(x)$ to a polynomial of maximum degree $m-1$?

Answer: Use modular reduction, similar to multiplication in $GF(p)$. For arithmetic in $GF(p^m)$ we need an *irreducible polynomial* of degree m with coefficients from $GF(p)$. Irreducible polynomials do not factor (except trivial factor involving 1) into smaller polynomials from $GF(p)$.

Example 1: $P(x) = x^4 + x + 1$ is irreducible over $GF(2)$ and can be used to construct $GF(2^4)$.

$C = A \cdot B \Rightarrow C(x) = A(x) \cdot B(x) \bmod P(x)$

$$
\begin{aligned}
A(x) &= x^3 + x^2 + 1 \\
B(x) &= x^2 + x \\
C'(x) &= A(x) \cdot B(x) = (x^5 + x^4 + x^2) + (x^4 + x^3 + x) = x^5 + x^3 + x^2 + 1
\end{aligned}
$$

$$\begin{aligned}
x^4 &= 1 \cdot P(x) + (x+1) \\
x^4 &\equiv x+1 \bmod P(x) \\
x^5 &\equiv x^2 + x \bmod P(x) \\
C(x) &\equiv C'(x) \bmod P(x) \\
C(x) &\equiv (x^2+x) + (x^3+x^2+1) = x^3 \\
A(x) \cdot B(x) &\equiv x^3
\end{aligned}$$

Note: in a typical computer representation, the multiplication would assign the following unusually looking operations:

$$\begin{array}{ccccc}
A & \cdot & B & = & C \\
(1\,1\,0\,1) & \cdot & (0\,1\,1\,0) & = & (1\,0\,0\,0)
\end{array}$$

Example 2: $x^4 + x^3 + x + 1$ is reducible since $x^4 + x^3 + x + 1 = (x^2+x+1)(x^2+1)$.

4. **Inversion in $GF(p^m)$:** the inverse A^{-1} of $A \in GF(p^m)^*$ is defined as:
   $A^{-1}(x) \cdot A(x) = 1 \bmod P(x)$
   $\Rightarrow$ perform the Extended Euclidean Algorithm with $A(x)$ and $P(x)$ as inputs

$$\begin{aligned}
s(x)P(x) + t(x)A(x) &= \gcd(P(x), A(x)) = 1 \\
\Rightarrow t(x)A(x) &= 1 \bmod P(x) \\
\Rightarrow t(x) = A^{-1}(x) &
\end{aligned}$$

Example: Inverse of $x^2 \in GF(2^3)$, with $P(x) = x^3 + x + 1$

$$\begin{array}{ll}
 & t_0 = 0,\ t_1 = 1 \\
x^3 + x + 1 = [x]x^2 + [x+1] & t_2 = t_0 - q_1 t_1 = -q_1 = -x = x \\
x + 1 = [1]x + 1 & t_3 = t_1 - q_2 t_2 = 1 - q_2 x = 1 - x = x + 1 \\
x = [x]1 + 0 & \\
\Rightarrow (x^2)^{-1} = t(x) = t_3 = x + 1 &
\end{array}$$

Check: $(x+1)x^2 = x^3 + x = (x+1) + x \equiv 1 \bmod P(x)$ since $x^3 \equiv x + 1 \bmod P(x)$.

Remark: In every iteration of the Euclidean algorithm, you should use long division (not shown above) to uniquely determine q_i and r_i.

5.4 Internal Structure

In the following, we assume a block length of 128 bits. The ShiftRow Sublayer works slightly differently for other block sizes.

5.4.1 Byte Substitution Layer

- Splits the incoming 128 bits in $128/8 = 16$ bytes.
- Each byte A is considered an element of $GF(2^8)$ and undergoes the following substitution individually

  1. $B = A^{-1} \in GF(2^8)$ where $P(x) = x^8 + x^4 + x^3 + x + 1$
  2. Apply affine transformation defined by:

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

  where $(b_7 \cdots b_0)$ is the vector representation of $B(x) = A^{-1}(x)$.

- The vector $C = (c_7 \cdots c_0)$ (representing the field element $c_7x^7 + \cdots + c_1x + c_0$) is the result of the substitution:

$$C = \text{ByteSub}(A)$$

  The entire substitution can be realized as a look-up in a 256×8-bit table with fixed entries.

  Remark: Unlike DES, Rijndael applies the same S-Box to each byte.

5.4.2 Diffusion Layer

- Unlike the non-linear substitution layer, the diffusion layer performs a linear operation on input words A, B. That means:

$$\text{DIFF}(A) \oplus \text{DIFF}(B) = \text{DIFF}(A + B)$$

- The diffusion layer consists of two sublayers.

ShiftRow SubLayer

1. Write an input word A as $128/8 = 16$ bytes and order them in a square array:
   Input $A = (a_0, a_1, \cdots, a_{15})$

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

2. Shift cyclically row-wise as follows:

$$\begin{array}{|c|c|c|c|}\hline a_0 & a_4 & a_8 & a_{12} \\ \hline a_5 & a_9 & a_{13} & a_1 \\ \hline a_{10} & a_{14} & a_2 & a_6 \\ \hline a_{15} & a_3 & a_7 & a_{11} \\ \hline \end{array} \quad \begin{array}{ll} & \text{0 positions} \\ ---\longrightarrow & \text{3 positions right shift} \\ --\longrightarrow & \text{2 positions right shift} \\ -\longrightarrow & \text{1 position right shift} \end{array}$$

MixColumn SubLayer

Principle: each column of 4 bytes is individually transformed into another column.

Question: How?

Each 4-byte column is considered as a vector and multiplied by a 4×4 matrix. The matirx contains *constant* entries. Multiplication and addition of the coefficients is done in $GF(2^8)$.

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Remarks:

1. Each c_i, b_i is an 8-bit value representing an element from $GF(2^8)$.

2. The small values $\{01, 02, 03\}$ allow for a very efficient implementation of the coefficient multiplication in the matrix. In software implementations, multiplication by 02 and 03 can be done through table look-up in a 256-by-8 table.

3. Additions in the vector-matrix multiplication are XORs.

5.4.3 Key Addition Layer

Simple bitwise XOR with a 128-bit subkey.

5.5 Decryption

Unlike DES and other Feistel ciphers, all of Rijndael layers must actually be inverted.

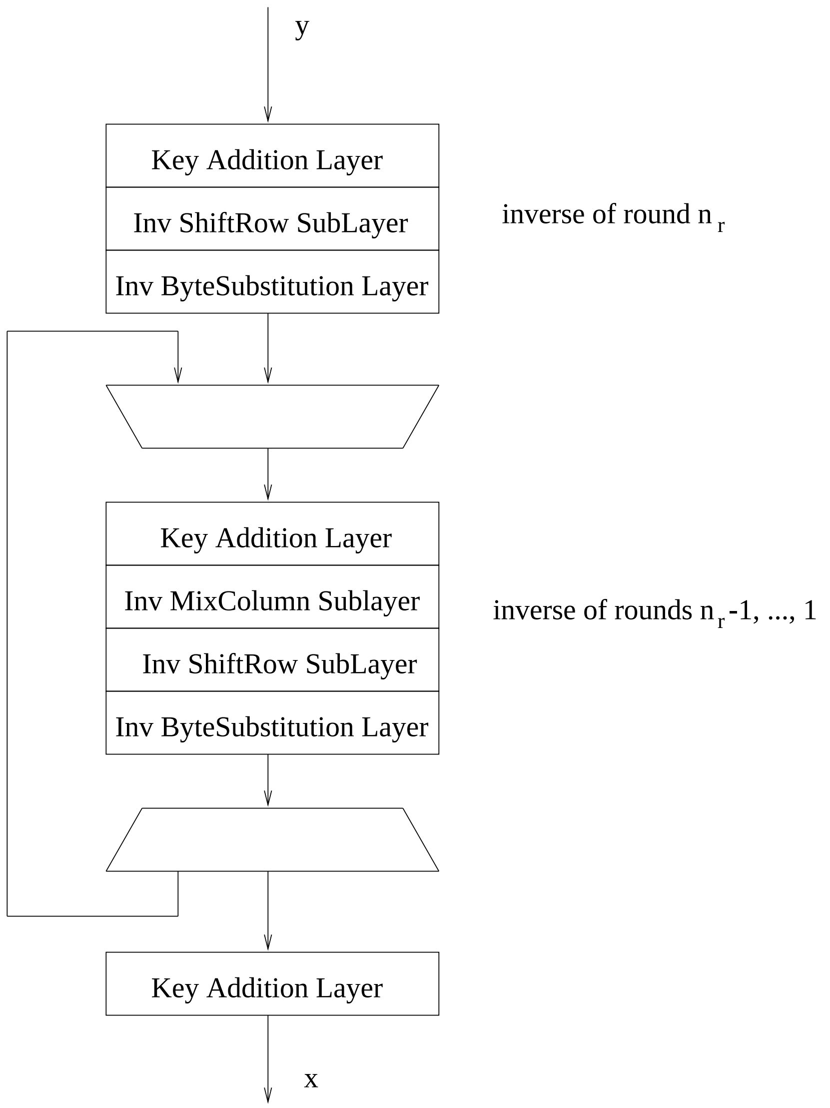


Figure 5.3: Rijndael decryption block diagram

Chapter 6

More about Block Ciphers

Further Reading:

Section 8.1 in [Sch93].

Note:

The following modes are applicable to all block ciphers $e_k(X)$.

6.1 Modes of Operation

6.1.1 Electronic Codebook Mode (ECB)

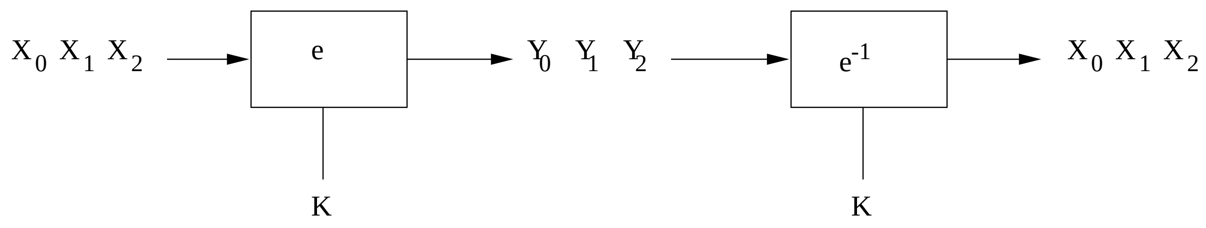


Figure 6.1: ECB model

General Description:

$e_k^{-1}(Y_i) = e_k^{-1}(e_k(X_i)) = X_i$; where the encryption can, for instance, be DES.

Problem:

This mode is susceptible to substitution attack because same X_i are mapped to same Y_i.

Example: Bank transfer.

Block #	1	2	3	4	5
	Sending Bank A	Sending Account #	Receiving Bank B	Receiving Account #	Amount $

Figure 6.2: ECB example

1. Tap encrypted line to bank B.
2. Send \$1.00 transfer to own account at bank B repeatedly $\rightarrow$ block 4 can be identified and recorded.
3. Replace in all messages to bank B block 4.
4. Withdraw money and fly to Paraguay.

Note: This attack is possible only for single-block transmission.

6.1.2 Cipher Block Chaining Mode (CBC)

Beginning: $Y_0 = e_k(X_0 \oplus IV)$.

$$X_0 = IV \oplus e_k^{-1}(Y_0) = IV \oplus e_k^{-1}(e_k(X_0 \oplus IV)) = X_0.$$

Encryption: $Y_i = e_k(X_i \oplus Y_{i-1})$.

Decryption: $X_i = e_k^{-1}(Y_i) \oplus Y_{i-1}$.

Question: How does it work?

$X_i = e_k^{-1}(e_k(X_i \oplus Y_{i-1})) \oplus Y_{i-1}$.

$X_i = (X_i \oplus Y_{i-1}) \oplus Y_{i-1}$.

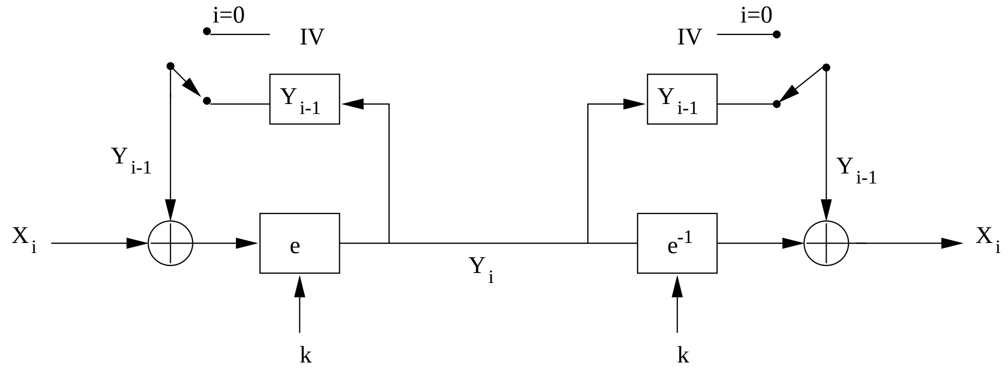


Figure 6.3: CBC model

$X_i = X_i. \quad q.e.d.$

Remark: The Initial Vector (IV) can be transmitted initially in cleartext.

6.1.3 Cipher Feedback Mode (CFB)

Assumption: block cipher with b bits block width and message with block width l, $1 \leq l \leq b$.

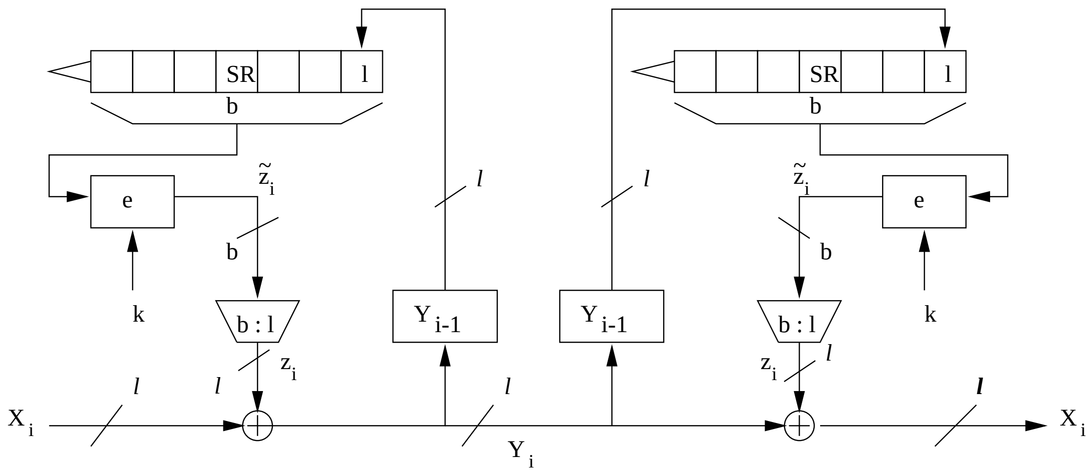


Figure 6.4: CFB model

Procedure:

1. Load shift register with initial value IV.
2. Encrypt $e_k(IV) = \tilde{z}_0$.
3. Take l leftmost bits: $\tilde{z}_0 \rightarrow z_0$.
4. Encrypt data: $Y_0 = X_0 \oplus z_0$.
5. Shift the shift register and load Y_0 into the rightmost SR position.
6. Go back to (2) substituting $e(IV)$ with $e(SR)$.

6.1.4 Counter Mode

Notes:

- Another mode which uses a block cipher as a pseudo-random generator.
- Counter Mode does not rely on previous ciphertext for encrypting the next block.
  $\Rightarrow$ well suited for parallel (hardware) implementation, with several encryption blocks working in parallel.
- Counter Mode stems from the Security Group of the ATM Forum, where high data rates required parallelization of the encryption process.

Description of Counter Mode:

1. An n-bit initial vector (IV) is loaded into a (maximum length) LFSR. The IV can be publically known, although a secret IV (i.e., the IV is considered part of the private key) turns the counter mode systems into a non-deterministic cipher which makes cryptoanalysis harder.
2. Encrypt block cipher input.

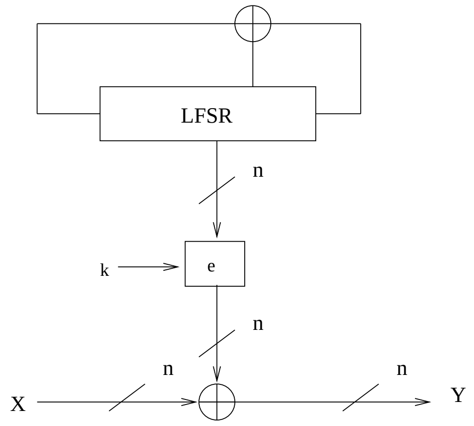


Figure 6.5: Counter Mode model

3. The block cipher output is considered a pseudorandom mask which is XORed with the plaintext.

4. The LFSR is clocked once (note: **all** input bits of the block cipher are shifted by one position).

5. Goto to Step 2.

Note that the period of a counter mode is $n \cdot 2^n$ which is very large for modern block ciphers, e.g., $128 \cdot 2^{128} = 2^{135}$ for AES algorithms.

6.2 Key Whitening

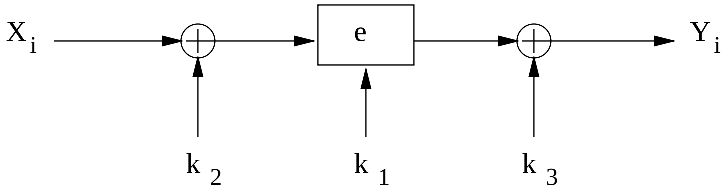


Figure 6.6: Whitening example

Encryption: $Y = e_{k_1,k_2,k_3}(X) = e_{k_1}(X \oplus k_2) \oplus k_3$.

Decryption: $X = e_{k_1}^{-1}(Y \oplus k_3) \oplus k_2$.

popular example: DESX

6.3 Multiple Encryption

6.3.1 Double Encryption

Note: The keyspace of this encryption is $|k| = 2^k \cdot 2^k = 2^{2k}$.

However, using the meet-in-the-middle attack, the key search is reduced significantly.

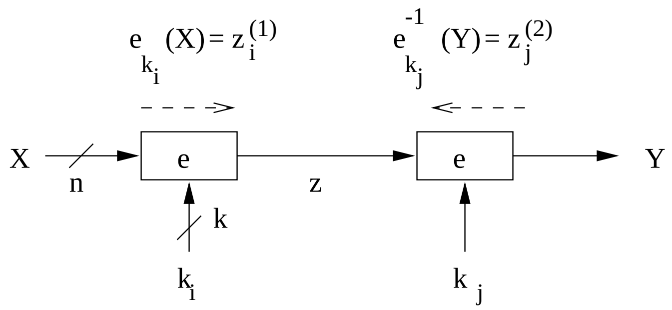


Figure 6.7: Double encryption and meet-in-the-middle attack

Meet in the middle attack:

Input $\rightarrow$ some pairs (x', y'), (x'', y''), $\ldots$.

Idea $\rightarrow$ compute $z_i^{(1)} = e_{k_i}(x')$ and $z_j^{(2)} = e_{k_j}^{-1}(y')$.

Problem $\rightarrow$ to find a matching pair such that $z_i^{(1)} = z_j^{(2)}$.

Procedure:

1. Compute a look-up table for all $(z_i^{(1)}, k_i)$, $i = 1, 2, \ldots, 2^k$ and store it in memory. Number of entries in the table is 2^k with each entry being n bits wide.

2. Find matching $z_j^{(2)}$.

   (a) compute $e_{k_j}^{-1}(y') = z_j^{(2)}$

   (b) if $z_j^{(2)}$ is in the look-up table, i.e., if $z_i^{(1)} = z_j^{(2)}$, check a few other pairs $(x'', y''), (x''', y'''), \ldots$ for the current keys k_i and k_j

   (c) if k_i and k_j give matching encryptions stop; otherwise go back to (a) and try different key k_j.

Question: How many additional pairs $(x'', y''), (x''', y'''), \ldots$ should we test?

General system: l subsequent encryptions and t pairs $(x', y'), (x'', y''), \ldots$.

1. In the first step there are 2^{lk} possible key combinations for the mapping $E(x') = e(\cdots(e(e(x'))\cdots) = y'$ but only 2^n possible values for x' and y'. Hence, there are

$$\frac{2^{lk}}{2^n}$$

mappings $E(x') = y'$. Note that only one mapping is done by the correct key!

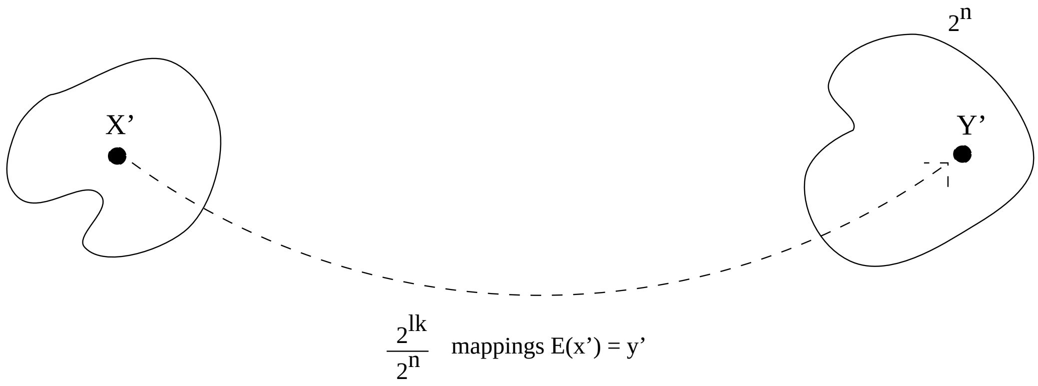


Figure 6.8: Number of mappings x' to y' under l-fold encryption

2. We use now a candidate key from step 1 and check whether $E(x'') = y''$. There are 2^n possible outcomes y for the mapping $E(x'')$. If a random key is used, the likelyhood that $E(x'') = y''$ is
$$\frac{1}{2^n}$$
If we check additionally a third pair (x''', y''') under the same "random" key from step 1, the likelyhood that $E(x'') = y''$ *and* $E(x''') = y'''$ is
$$\frac{1}{2^{2n}}$$
If we check $t-1$ additional pairs $(x'', y''), (x''', y'''), \ldots (x^{(t)}, y^{(t)})$ the likelyhood that a random key fulfills $E(x'') = y''$, $E(x''') = y''', \ldots$ is
$$\frac{1}{2^{(t-1)n}}$$

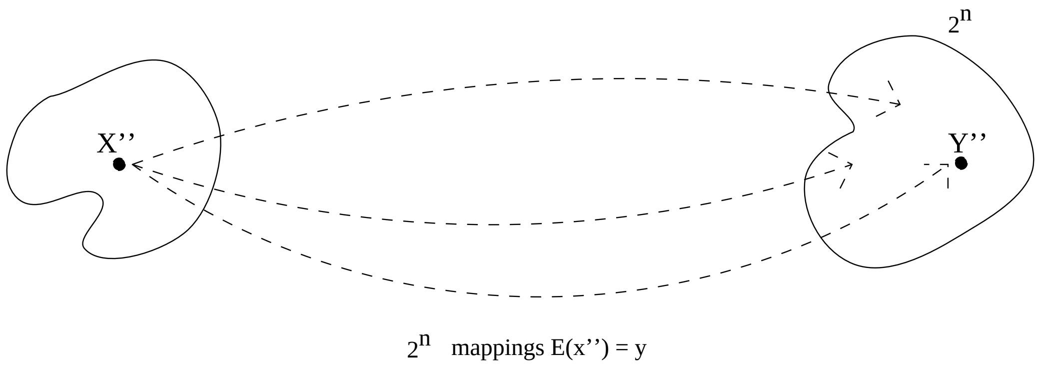


Figure 6.9: Number of mappings x'' to y

3. Since there are $\frac{2^{lk}}{2^n}$ candidate keys in step 1, the likelyhood that at least one of the candidate keys fulfills all $E(x'') = y''$, $E(x''') = y''', \ldots$ is
$$\frac{1}{2^{(t-1)n}} \frac{2^{lk}}{2^n} = 2^{lk-tn}$$

Example: Double encryption with DES. We use two pairs $(x', y'), (x'', y'')$. The likelyhood that an incorrect key pair k_i, k_j is picked is
$$2^{lk-tn} = 2^{112-128} = 2^{-16}$$

If we use three pairs $(x', y'), (x'', y''), (x''', y''')$, the likelyhood that an incorrect key pair k_i, k_j is picked is

$$2^{lk-tn} = 2^{112-192} = 2^{-80}$$

Computational complexity:

Brute force attack: 2^{2k}.

Meet in the middle attack: 2^k encryptions + 2^k decryptions = 2^{k+1} computations and 2^k memory locations.

6.3.2 Triple Encryption

Option 1:

$Y = e_{k_1}(e_{k_2}^{-1}(e_{k_1}(X)))$; if $k_1 = k_2 \rightarrow Y = e_{k_1}(X)$.

Option 2:

$Y = e_{k_3}(e_{k_2}(e_{k_1}(X)))$; where $|k| \approx 2^{2k}$

Option 2 should be preferred.

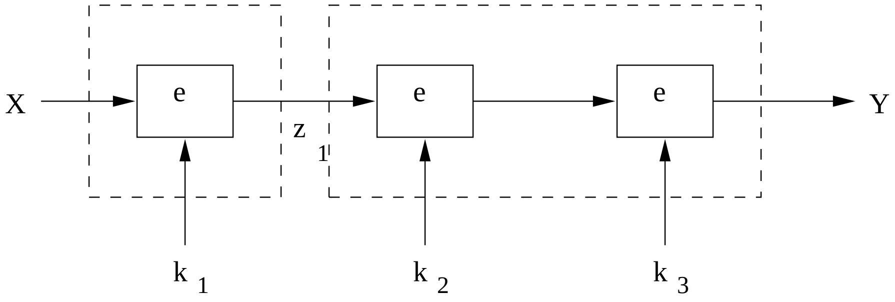


Figure 6.10: Triple encryption example

Note:

Meet in the middle attack can be used in a similar way by storing z_i results in memory. The computational complexity of this approach is $2^k \cdot 2^k = 2^{2k}$.

Chapter 7

Introduction to Public-Key Cryptography

7.1 Principle

Quick review of private-key cryptography

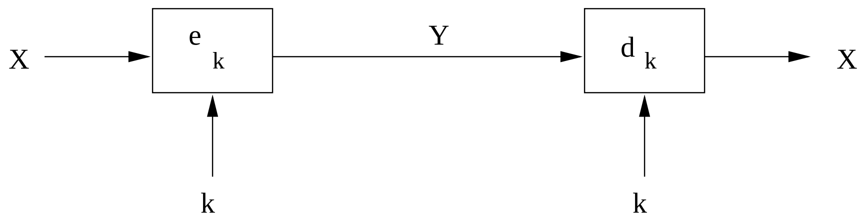


Figure 7.1: Private-key model

Two properties of private-key schemes:

1. The algorithm requires same secret key for encryption and decryption.

2. Encryption and decryption are essentially identical (symmetric algorithms).

Analogy for private key algorithms

Private key schemes are analogous to a safe box with a strong lock. Everyone with the key can deposit messages in it and retrieve messages.

Main problems with private key schemes are:

1. Requires secure transmission of secret key.

2. In a network environment, each pair of users has to have a different key resulting in too many keys ($n \cdot\ (n-1) \div 2$ key pairs).

New Idea:

Make a slot in the safe box so that everyone can deposit a message, but only the receiver can open the safe and look at the content of it. This idea was proposed in [WD76] in 1976 by Diffie/Hellman.

Idea: Split key.

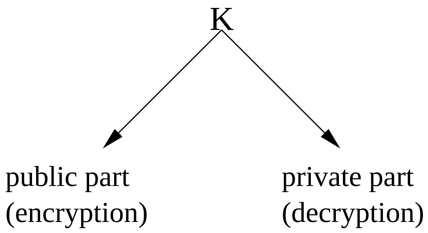


Figure 7.2: Split key idea

Protocol:

1. Alice and Bob agree on a public-key cryptosystem.
2. Bob sends Alice his public key.
3. Alice encrypts her message with Bob's public key and sends the ciphertext.
4. Bob decrypts ciphertext using his private key.

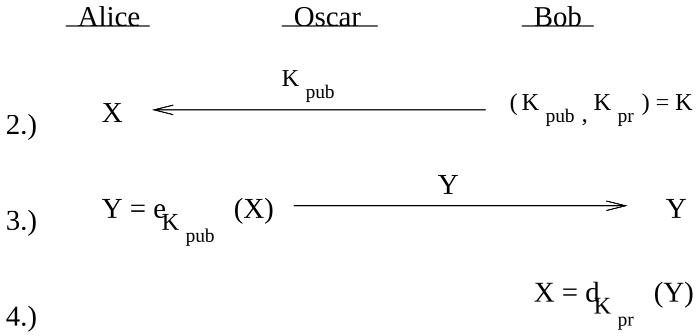


Figure 7.3: Public-key encryption protocol

7.2 One-Way Functions

All public-key algorithms are based on one-way functions.

Definition 7.2.1 *A function* f *is a "one-way function" if:*

(a) $y = f(x) \rightarrow$ *is easy to compute,*

(b) $x = f^{-1}(y) \rightarrow$ *is very hard to compute.*

Example: Discrete Logarithm (DL) one-way Function

$2^x \bmod 127 \equiv 31$

$x = ?$

Definition 7.2.2 *A* trapdoor one function *is a one-way function whose inverse is easy to compute given a side information such as the private key.*

7.3 Overview of Public-Key Algorithms

There are three families of Public-Key (PK) algorithms of practical relevance:

1. Integer factorization algorithms (RSA, ...)

2. Discrete logarithms (D–H, DSA, ...)

3. Elliptic curves (EC)

⇒ Generally speaking, public-key algorithms are much slower than private-key algorithms.

⇒ Public-Key algorithms are mainly used for key establishment and digital signatures and **not** for bulk data encryption.

Algorithm Family	Bit length of the operands
Integer Factorization (RSA)	1024
Discrete Logarithm (D–H, DSA)	1024
Elliptic curves	160
Block cipher	80

Table 7.1: Bit lengths for security level of approximately 2^{80} computations for successful attack.

7.4 Important Public-Key Standards

a) IEEE P1363. Comprehensive standard of public-key algorithms. Collection of IF, DL, and EC algorithm families, including in particular:

   – Key establishment algorithms

   – Key transport algorithms

   – Signature algorithms

   Note: IEEE P1363 does not recommend any bit lengths or security levels.

b) ANSI Banking Security standards.

ANSI#	Subject
X9.30–1	digital signature algorithm (DSA)
X9.30–2	hashing algorithm for RSA
X9.31–1	RSA signature algorithm
X9.32–2	hashing algorithms for RSA
X9.42	key management using Diffe-Hellman
X9.62 (draft)	elliptic curve digital signature algorithm (ECDSA)
X9.63 (draft)	elliptic curve key agreement and transport protocols

c) U.S. Government standards (FIPS)

FIPS#	Subject
FIPS 180-1	secure hash standard (SHA-1)
FIPS 186	digital signature standard (DSA)
FIPS JJJ (draft)	entity authentication (asymetric)

7.5 More Number Theory

7.5.1 Euclid's Algorithm

Basic Form

Given r_0 and r_1 with one larger than the other, compute the $\gcd(r_0, r_1)$.

Example 1:

$r_0 = 22, r_1 = 6.$

$\gcd(r_0, r_1) = ?$

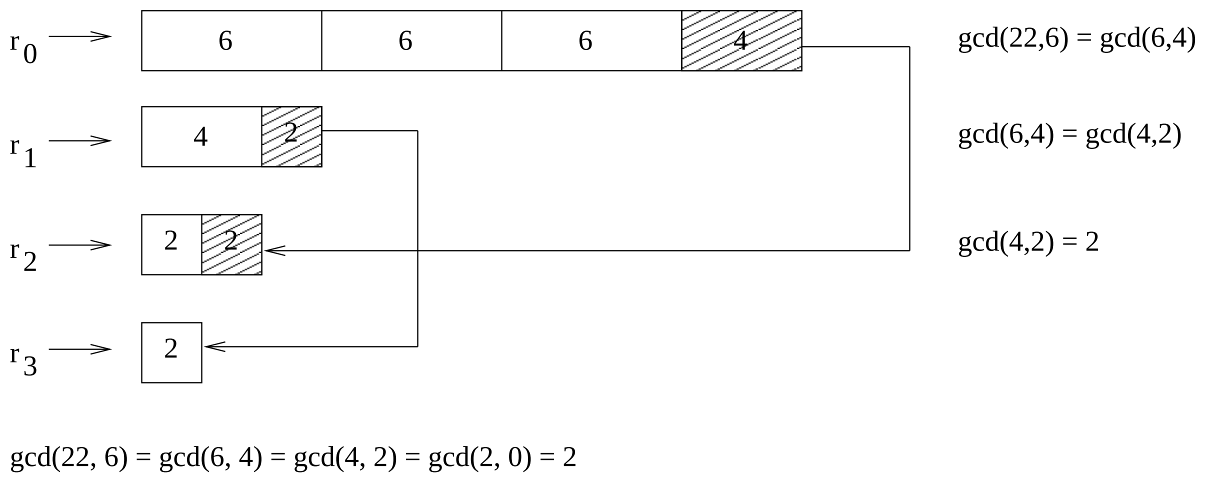


Figure 7.4: Euclid's algorithm example

Example 2:

$r_0 = 973;\ r_1 = 301.$

$973 = 3 \cdot 301 + 70.$

$301 = 4 \cdot 70 + 21.$

$70 = 3 \cdot 21 + 7.$

$21 = 3 \cdot 7 + 0.$

$\gcd(973, 301) = \gcd(301, 70) = \gcd(70, 21) = \gcd(21, 7) = 7.$

Algorithm:

$$
\begin{array}{ll}
\text{input: } r_0,\, r_1 & \\
r_0 = q_1 \cdot r_1 + r_2 & \gcd(r_0, r_1) = \gcd(r_1, r_2) \\
r_1 = q_2 \cdot r_2 + r_3 & \gcd(r_1, r_2) = \gcd(r_2, r_3) \\
\vdots & \vdots \\
r_{m-2} = q_{m-1} \cdot r_{m-1} + r_m & \gcd(r_{m-2}, r_{m-1}) = \gcd(r_{m-1}, r_m) \\
r_{m-1} = q_m \cdot r_m + 0 \leftarrow \dagger & \gcd(r_0, r_1) = \gcd(r_{m-1}, r_m) = r_m
\end{array}
$$

$\dagger$ - termination criteria

Extended Euclidean Algorithm

Theorem 7.5.1 *Given two integers r_0 and r_1, there exist two other integers s and t such that $s \cdot r_0 + t \cdot r_1 = \gcd(r_0, r_1)$.*

Question: How to find s and t?

Use Euclid's algorithm and express the current remainder r_i in every iteration in the form $r_i = s_i r_0 + t_i r_1$. Note that in the last iteration $r_m = \gcd(r_0, r_1) \stackrel{!}{=} s_m r_0 + t_m r_1 = s r_0 + t r_1$.

index	Euclid's Algorithm	$r_j = s_j \cdot r_0 + t_j \cdot r_1$
2	$r_0 = q_1 \cdot r_1 + r_2$	$r_2 = r_0 - q_1 \cdot r_1 = s_2 \cdot r_0 + t_2 \cdot r_1$
3	$r_1 = q_2 \cdot r_2 + r_3$	$r_3 = r_1 - q_2 \cdot r_2 = r_1 - q_2(r_0 - q_1 \cdot r_1)$
		$= [-q_2]r_0 + [1 + q_1 \cdot q_2]r_1 = s_3 \cdot r_0 + t_3 \cdot r_1$
$\vdots$	$\vdots$	$\vdots$
i	$r_{i-2} = q_{i-1} \cdot r_{i-1} + r_i$	$r_i = s_i \cdot r_0 + t_i \cdot r_1$
$i+1$	$r_{i-1} = q_i \cdot r_i + r_{i+1}$	$r_{i+1} = s_{i+1} \cdot r_0 + t_{i+1} \cdot r_1$
$i+2$	$r_i = q_{i+1} \cdot r_{i+1} + r_{i+2}$	$r_{i+2} = r_i - q_{i+1} \cdot r_{i+1}$
		$= (s_i \cdot r_0 + t_1 \cdot r_1) - q_{i+1}(s_{i+1} \cdot r_0 + t_{i+1} \cdot r_1)$
		$= [s_i - q_{i+1}] \cdot s_{i+1}]r_0 + [t_1 - q_{i+1} \cdot t_{i+1}]r_1$
		$= s_{i+2} \cdot r_0 + t_{i+2} \cdot r_1$
$\vdots$	$\vdots$	$\vdots$
m	$r_{m-2} = q_{m-1} \cdot r_{m-1} + r_m$	$r_m = \gcd(r_0, r_1) = s_m \cdot r_0 + t_m \cdot r_1$

Now: $s = s_m$, $t = t_m$

Recursive formulae:

$$\begin{array}{ll} s_0 = 1, & t_0 = 0 \\ s_1 = 0, & t_1 = 1 \\ s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}, & t_i = t_{i-2} - q_{i-1} \cdot t_{i-1};\ i = 2, 3, 4 \ldots \end{array}$$

Remark:

a) Extended Euclidean algorithm is commonly used to compute the inverse element in Z_m. If $\gcd(r_0, r_1) = 1$, then $t = r_1^{-1} \bmod r_0$.

b) For fast software implementation, the "binary extended Euclidean algorithm" is more efficient [AM97] because it avoids the division required in each iteration of the extended Euclidean algorithm shown above.

7.5.2 Euler's Phi Function

Definition 7.5.1 *The number of integers in Z_m relatively prime to m is denoted by $\Phi(m)$.*

Example 1:

$m = 6$; $Z_6 = \{0, 1, 2, 3, 4, 5\}$

$\gcd(0, 6) = 6$

$\gcd(1, 6) = 1 \leftarrow$

$\gcd(2, 6) = 2$

$\gcd(3, 6) = 3$

$\gcd(4, 6) = 2$

$\gcd(5, 6) = 1 \leftarrow$

$\Phi(6) = 2$

Example 2:

$m = 5$; $Z_5 = \{0, 1, 2, 3, 4\}$

$\gcd(0, 5) = 5$

$\gcd(1, 5) = 1 \ \leftarrow$

$\gcd(2, 5) = 1 \ \leftarrow$

$\gcd(3, 5) = 1 \ \leftarrow$

$\gcd(4, 5) = 1 \ \leftarrow$

$\Phi(5) = 4$

> **Theorem 7.5.2** *If $m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$, where p_i are prime numbers and e_i are integers, then:*
>
> $$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1})$$
>
> .

Example:

$m = 40 = 8 \cdot 5 = 2^3 \cdot 5 = p_1^{e_1} \cdot p_2^{e_2}$

$\Phi(m) = (2^3 - 2^2)(5^1 - 5^0) = (8 - 4)(5 - 1) = 4 \cdot 4 = 16$

> **Theorem 7.5.3** Euler's Theorem
>
> *If* $\gcd(a, m) = 1$, *then:*
>
> $$a^{\Phi(m)} \equiv 1 \bmod m$$
>
> .

Example:

$m = 6$; $a = 5$

$\Phi(6) = \Phi(3 \cdot 2) = (3 - 1)(2 - 1) = 2$

$5^{\Phi(6)} = 5^2 = 25 \equiv 1 \bmod 6$

Chapter 8

RSA

1. Most popular public-key cryptosystem.

2. Invented by Rivest/Shamir/Adleman in 1977 at MIT.

3. Patented until 2000.

8.1 Cryptosystem

Set-up Stage

1. Choose two large primes p and q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Choose random b; $0 < b < \Phi(n)$, with $\gcd(b, \Phi(n)) = 1$.

   Note that b has inverse in $Z_{\Phi(n)}$.
5. Compute inverse $a = b^{-1} \bmod \Phi(n)$:

$$b \cdot a \equiv 1 \bmod \Phi(n).$$

6. Public key: $k_{pub} = (n, b)$.

   Private key: $k_{pr} = (p, q, a)$.

Encryption: done using public key, k_{pub}.

$y = e_{k_{pub}}(x) = x^b \bmod n$.

$x \in Z_n = \{0, 1, \ldots, n-1\}$.

Decryption: done using private key, k_{pr}.

$x = d_{k_{pr}}(y) = y^a \bmod n$.

Example:

Alice sends encrypted message $(x = 4)$ to Bob after Bob sends her the public key.

Alice		Bob
		(1) choose $p = 3$; $q = 11$
		(2) $n = p \cdot q = 33$
		(3) $\Phi(n) = (3-1)(11-1) = 2 \cdot 10 = 20$
		(4) choose $b = 3$; $\gcd(20, 3) = 1$
$x = 4$	$\xleftarrow{k_{pub}(3,33)}$	(5) $a = b^{-1} = 7 \bmod 20$
$y = x^b \bmod n = 4^3 = 64 \equiv 31 \bmod 33$	$\xrightarrow{y=31}$	$x = y^a = 31^7 \equiv 4 \bmod 33$

Why does RSA work?

We have to show that: $d_{k_{pr}}(y) = d_{k_{pr}}(e_{k_{pub}}(x)) = x$.

$d_{k_{pr}} = y^a = x^{ba} = x^{ab} \bmod n$.

$a \cdot b \equiv 1 \bmod \Phi(n) \Longleftrightarrow a \cdot b \equiv 1 + t \cdot \Phi(n)$; t is an integer.

$d_{k_{pr}} = x^{ab} = x^{t \cdot \Phi(n)} \cdot x^1 = (x^{\Phi(n)})^t \cdot x \bmod n$.

if $x^{\Phi(n)} \equiv 1 \bmod n$ then $d_{k_{pr}} = (x^{\Phi(n)})^t \cdot x = 1^t \cdot x = 1 \cdot x = x \bmod n$.

<u>1. Case:</u> $\gcd(x, n) = \gcd(x, p \cdot q) = 1$

Euler's Theorem: $x^{\Phi(n)} \equiv 1 \bmod n$, $\quad q.e.d.$

<u>2. Case:</u> $\gcd(x, n) = \gcd(x, p \cdot q) \neq 1$

either $x = r \cdot p$ or $x = s \cdot q$; r, s are integers such that; $r < q$, $s < p$.

assume $x = r \cdot p \Rightarrow \gcd(x, q) = 1$

$x^{\Phi(n)} = x^{(q-1)(p-1)} = x^{\Phi(q)(p-1)} = (x^{\Phi(q)})^{p-1} = 1 \bmod q$

$x^{\Phi(n)} = 1 + c \cdot q$; where c is an integer

$x \cdot x^{\Phi(n)} = x + x \cdot c \cdot q = x + r \cdot p \cdot c \cdot q = x + r \cdot c \cdot p \cdot q = x + r \cdot c \cdot n$

$x \cdot x^{\Phi(n)} \equiv x \bmod n$

$$x^{\Phi(n)} \equiv 1 \bmod n, \quad q.e.d.$$

8.2 Computational Aspects

8.2.1 Choosing p and q

Problem: Finding two large primes p, q (each > 250 bits).

Principle:

> Pick a large integer and apply primality test. In practice, a "Monte Carlo" test developed by Miller-Rabbin (pg. 136 in [Sti95]) is used. Note that a primality test does NOT require factorization.

Miller-Rabin Algorithm:

<u>Input</u>: p or q and arbitrary number $r < p, q$. <u>Output 1</u>: Statement "p, q is composite" $\rightarrow$ always true. <u>Output 2</u>: Statement "p, q is prime" $\rightarrow$ true with probability > 0.75.

In practice, the above algorithm is run 3 times (for a 1000 bit prime) and upto 12 times (for a 150 bit prime) [AM97, Table 4.4 page 148] with different parameters r. If the answer is always "p is prime", then p is with very high probability a prime.

$$\mathcal{P}(p \text{ is composite }) \leq 0.25^t \text{ where } t = \text{number of tries.}$$

Question: What is the likelihood that a randomly picked integer p or q is prime?
Answer: $\mathcal{P}(p \text{ is prime }) \approx \frac{1}{ln(p)}$.

Example: $p \approx 2^{250} \rightarrow$ (250 bits).

$$\mathcal{P}(p \text{ is prime }) = \frac{1}{\ln(2^{250})} \approx \frac{1}{173}.$$

8.2.2 Choosing a and b

$k_{pub} = b$; condition: $\gcd(b, \Phi(n)) = 1$; where $\Phi(n) = (p-1)\cdot(q-1)$.

$k_{pr} = a$; where $a = b^{-1} \bmod \Phi(n)$.

Pick arbitrary b (large!) and compute:

1. Euclidean Algorithm: $s \cdot \Phi(n) + t \cdot b = \gcd(b, \Phi(n))$
2. Test if $\gcd(b, \Phi(n)) = 1$
3. Calculate a:

   Question: What is $t \cdot b \bmod \Phi(n)$?

$$\begin{aligned} t\cdot b &= (-s)\Phi(n) + 1 \\ \Rightarrow t \cdot b &\equiv 1 \bmod \Phi(n) \\ \Rightarrow t &= b^{-1} = a \bmod \Phi(n) \end{aligned}$$

Remark:

It is not necessary to find s for the computation of a.

8.2.3 Encryption/Decryption

encryption: $e_{k_{pub}}(x) = x^b \bmod n = y$.

decryption: $d_{k_{pr}}(y) = y^a \bmod n = x$.

Question: How many multiplications are required for computing x^8?

Answer: $\underbrace{x \cdot x = x^2}_{1}$; $\underbrace{x^2 \cdot x^2 = x^4}_{2}$; $\underbrace{x^4 \cdot x^4 = x^8}_{3}$.

if $0 < b < \Phi(n)$ then $\mathcal{O}(\Phi(n)) \approx \mathcal{O}(n)$.

Question: How many multiplications are required for computing x^{13}?

Answer: $\underbrace{x \cdot x = x^2}_{\text{SQ}}$; $\underbrace{x^2 \cdot x = x^3}_{\text{MUL}}$; $\underbrace{x^3 \cdot x^3 = x^6}_{\text{SQ}}$; $\underbrace{x^6 \cdot x^6 = x^{12}}_{\text{SQ}}$; $\underbrace{x^{12} \cdot x = x^{13}}_{\text{MUL}}$.

Square-and-multiply algorithm

First: binary representation of the exponent $\rightarrow x^B$; $B \leq 15$

$B = b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0$

$B = (b_3 \cdot 2 + b_2)2^2 + b_1 \cdot 2 + b_0 = ((b_3 \cdot 2 + b_2)2 + b_1)2 + b_0$

$x^B = x^{((b_3 \cdot 2 + b_2)2 + b_1)2 + b_0}$

Step	x^B
#1	$x^{b_3 \cdot 2}$
#2	$(x^{b_3 \cdot 2} \cdot x^{b_2})$
#3	$(x^{b_3 \cdot 2} \cdot x^{b_2})^2$
#4	$(x^{b_3 \cdot 2} \cdot x^{b_2})^2 \cdot x^{b_1}$
#5	$((x^{b_3 \cdot 2} \cdot x^{b_2})^2 \cdot x^{b_1})^2$
#6	$((x^{b_3 \cdot 2} \cdot x^{b_2})^2 \cdot x^{b_1})^2 \cdot x^{b_0}$

Example: $x^{13} = x^{1101_2} = x^{(b_3, b_2, b_1, b_0)_2}$

#1	$x^{b_3 \cdot 2} = x^2$	SQ
#2	$x^2 \cdot x^{b_3} = x^2 \cdot x = x^3$	MUL
#3	$(x^3)^2 = x^6$	SQ
#4	$x^6 \cdot x^0 \; x^6 \cdot 1 = x^6$	
#5	$(x^6)^2 = x^{12}$	SQ
#6	$x^{12} \cdot x^{b_0} = x^{12} \cdot x = x^{13}$	MUL

Complexity: $[\log_2 n] \cdot \mathrm{SQ} + [\frac{1}{2} \log_2 n] \cdot \mathrm{MUL}$.

Comparison: $B = 2^{1000}$

Straight forward exponentiation: $2^{1000} \approx 10^{300}$ multiplications

$\rightarrow$ computationally impossible.

Square-and-multiply: $1.5 \cdot \log_2(2^{1000}) = 1500$ multiplications and squarings

$\rightarrow$ relatively easy.

Remark: Remember to apply modulo reduction after every multiplication and squaring operation.

Algorithm [Sti95]: computes x^B, where $B = \sum_{i=0}^{l-1} b_i 2^i$

1. $z = x$
2. for $i = l-1$ downto 0 do:
   (a) $z = z^2 \bmod n$
   (b) if $(b_i = 1)$ then $z = z \cdot x \bmod n$

8.3 Attacks

8.3.1 Brute Force

Given $y = x^b \bmod n$, try all possible keys a; $0 \le a < \Phi(n)$ to obtain $x = y^a \bmod n$. In practice $|\mathcal{K}| = \Phi(n) \approx n > 2^{500} \Rightarrow$ impossible.

8.3.2 Finding $\Phi(n)$

Given $n, b, y = x^b \bmod n$, find $\Phi(n)$ and compute $a = b^{-1} \bmod \Phi(n)$.
$\Rightarrow$ computing $\Phi(n)$ is believed to be as difficult as factoring n.

8.3.3 Finding a directly

Given $n, b, y = x^b \bmod n$, find a directly and compute $x = y^a \bmod n$.
$\Rightarrow$ computing a directly is believed to be as difficult as factoring n.

8.3.4 Factorization of n

Given $n, b, y = x^b \bmod n$, find $p \cdot q = n$ and compute:

$\Phi(n) = (p-1)(q-1)$

$b = a^{-1} \bmod \Phi(n)$

$x = y^a \bmod n$

$\rightarrow$ This approach is the only attack believed to be practical.

Factoring Algorithms:

1. Quadratic Sieve (QS): speed depends on the size of n; record: in 1994 factoring of $n =$RSA129, $\log_{10} n = 129$ digits, $\log_2 n = 426$ bits.

2. Elliptic Curve: similar to QS; speed depends on the size of the smallest prime factor of n, i.e., on p and q.

3. Number Field Sieve: asymptotically better than QS; record: in 1996 factoring of $n =$RSA140; $\log_{10} n = 140$ digits; $\log_2 n = 466$ bits.

Algorithm	*Complexity*
Quadratic Sieve	$\mathcal{O}(e^{(1+o(1))\sqrt{\ln(n)\ln(\ln(n))}})$
Elliptic Curve	$\mathcal{O}(e^{(1+o(1))\sqrt{2\ln(p)\ln(\ln(p))}})$
Number Field Sieve	$\mathcal{O}(e^{(1.92+o(1))(\ln(n))^{1/3}(\ln(\ln(n)))^{2/3}})$

number	month	MIPS-years	algorithm
RSA-100	April 1991	7	quadratic sieve
RSA-110	April 1992	75	quadratic sieve
RSA-120	June 1993	830	quadratic sieve
RSA-129	April 1994	5000	quadratic sieve
RSA-130	April 1996	500	generalized number field sieve
RSA-140	February 1999	1500	generalized number field sieve
RSA-155	August 1999	8000	generalized number field sieve

8.4 Implementation

- Hardware: 1024 bit decryption in less that 5 ms.
- Software: 1024 bit decryption in 43 ms; 1024 bit encryption in 0.65 ms
- hybrid systems, consisting of public-key and private-key algorithms: most commonly used in practice
  1. key exchange and authentication with (slow) public-key algorithm
  2. bulk data encryption with (fast) block ciphers

Chapter 9

The Discrete Logarithm (DL) Problem

- DL is the underlying one-way function for:
  1. Diffie-Hellman key exchange.
  2. DSA (digital signature algorithm).
  3. ElGamal encryption/digital signature scheme.
  4. Elliptic curve cryptosystems.
  5. ......
- DL is based on finite groups.

9.1 Some Algebra

Further Reading: [Big85].

9.1.1 Groups

Definition 9.1.1 *A <u>group</u> is a set $\mathcal{G}$ of elements together with a binary operation "o" such that:*

1. *If $a, b \in \mathcal{G}$ then $a \circ b = c \in \mathcal{G} \rightarrow$ (closure).*
2. *If $(a \circ b) \circ c = a \circ (b \circ c) \rightarrow$ (associativity).*
3. *There exists an identity element $e \in \mathcal{G}$:*
   $e \circ a = a \circ e = a \rightarrow$ *(identity).*
4. *There exists an inverse element $\tilde{a}$, for all $a \in \mathcal{G}$:*
   $a \circ \tilde{a} = e \rightarrow$ *(inverse).*

Examples:

1. $\mathcal{G}= Z = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
   $\circ =$ addition
   $(Z, +)$ is a group with $e = 0$ and $\tilde{a} = -a$
2. $\mathcal{G}= Z$
   $\circ =$ multiplication
   $(Z, \times)$ is NOT a group since inverses $\tilde{a}$ do not exist except for $a = 1$
3. $\mathcal{G}=\mathcal{C}$ (complex numbers $u + iv$)
   $\circ =$ multiplication
   $(\mathcal{C}, \times)$ is a group with $e = 1$ and

$$\tilde{a} = a^{-1} = \frac{u - iv}{u^2 + v^2}$$

Definition 9.1.2 *"Z_n^*" denotes the set of numbers i, $0 \leq i < n$, which are relatively prime to n.*

Examples:

1. $Z_9^* = \{1, 2, 4, 5, 7, 8\}$

2. $Z_7^* = \{1, 2, 3, 4, 5, 6\}$

Multiplication Table

$*$ mod 9	1	2	4	5	7	8
1	1	2	4	5	7	8
2	2	4	8	1	5	7
4	4	8	7	2	1	5
5	5	1	2	7	8	4
7	7	5	1	8	4	2
8	8	7	5	4	2	1

Theorem 9.1.1 *Z_n^* forms a group under modulo n multiplication. The identity element is $e = 1$.*

Remark:

The inverse of $a \in Z_n^*$ can be found through the extended Euclidean algorithm.

9.1.2 Finite Groups

Definition 9.1.3 *A group $(\mathcal{G}, \circ)$ is* ***finite*** *if it has a finite number of g elements. We denote the cardinality of $\mathcal{G}$ by $|\mathcal{G}|$.*

Examples:

1. $(Z_m, +)$: $a + b = c \bmod m$

   Question: What is the cardinality $\rightarrow |Z_m| = m$

   $Z_m = \{0, 1, 2, \ldots, m - 1\}$

2. $(Z_p^*, \times)$: $a \times b = c \bmod p$; p is prime

   Question: What is the cardinality $\rightarrow |Z_p^*| = p - 1$

   $Z_p^* = \{1, 2, \ldots, p-1\}$

> **Definition 9.1.4** *The **order** of an element $a \in (\mathcal{G}, \circ)$ is the smallest positive integer o such that $a \circ a \circ \ldots \circ a = a^o = 1$.*

Example: $(Z_{11}^*, \times)$, $a = 3$

Question: What is the order of $a = 3$?

$a^1 = 3$

$a^2 = 3^2 = 9$

$a^3 = 3^3 = 27 \equiv 5 \bmod 11$

$a^4 = 3^4 = 3^3 \cdot 3 = 5 \cdot 3 = 15 \equiv 4 \bmod 11$

$a^5 = a^4 \cdot a = 4 \cdot 3 = 12 \equiv 1 \bmod 11$

$\Rightarrow \mathrm{ord}(3) = 5$

Definition 9.1.5 *A group $\mathcal{G}$ which contains elements α with maximum order $ord(\alpha) = |\mathcal{G}|$ is said to be **cyclic**. Elements with maximum order are called **generators** or **primitive elements**.*

Example: 2 is a primitive element in Z_{11}^*

$|Z_{11}^*| = |\{1,2,3,4,5,6,7,8,9,10\}| = 10$

$a = 2$

$a^2 = 4$

$a^3 = 8$

$a^4 = 16 \equiv 5$

$a^5 = 10;$

$a^6 = 20 \equiv 9$

$a^7 = 18 \equiv 7$

$a^8 = 14 \equiv 3;$

$a^9 = 6$

$a^{10} = 12 \equiv 1$

$\underline{a^{11} = 2 = a}.$

$\Rightarrow \text{ord}(a = 2) = 10 = |Z_{11}^*|$

$\Rightarrow$ (1) $|Z_{11}^*|$ is cyclic

$\Rightarrow$ (2) $a = 2$ is a primitive element

Observation (important): 2^i; $i = 1, 2, \ldots, 10$ **generates** all elements of Z_{11}^*

i	1	2	3	4	5	6	7	8	9	10
2^i	2	4	8	5	10	9	7	3	6	1

Some properties of cyclic groups:

1. The number of primitive elements is $\Phi(|\mathcal{G}|)$.
2. For every $a \in \mathcal{G}$: $a^{|\mathcal{G}|} = 1$.
3. For every $a \in \mathcal{G}$: $\text{ord}(a)$ divides $|\mathcal{G}|$.

Proof only for (2): $a = \alpha^i$

$a^{|\mathcal{G}|} = (\alpha^i)^{|\mathcal{G}|} = (\alpha^{|\mathcal{G}|})^i \doteq 1^i = 1.$

Example: Z_{11}^*; $|Z_{11}^*| = 10$

1. $\Phi(10) = (2-1)(5-1) = 1 \cdot 4 = 4$
2. $a = 3 \rightarrow 3^{10} = (3^5)^2 = 1^2 = 1$
3. homework ...

9.2 The General DL Problem

Given a cyclic subgroup $(\mathcal{G}, \circ)$ and a primitive element α. Let

$$\beta = \underbrace{\alpha \circ \alpha \ldots \alpha}_{i \quad times} = \alpha^i$$

be an arbitrary element in $\mathcal{G}$.

General DL Problem:

$$\boxed{\begin{array}{l}\text{Given } \mathcal{G},\ \alpha, \beta = \alpha^i, \text{ find } i. \\ \\ \qquad\qquad i = \log_\alpha(\beta)\end{array}}$$

Examples:

1. $(Z_{11}, +)$; $\alpha = 2$; $\beta = \underbrace{2 + 2 + \ldots + 2}_{i \quad times} = i \cdot 2$

i	1	2	3	4	5	6	7	8	9	10	11
2i	2	4	6	8	10	1	3	5	7	9	0

   Let $i = 7$: $\beta = 7 \cdot 2 \equiv 3 \bmod 11$

   Question: given $\alpha = 2$, $\beta = 3 = i \cdot 2$, find i

   Answer: $i = 2^{-1} \cdot 3 \bmod 11$

   Euclid's algorithm can be used to compute i thus this example is NOT a one-way function.

2. $(Z_{11}^*, \times)$; $\alpha = 2$; $\beta = \underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{i \quad times} = 2^i$

   $\beta = 3 = 2^i \bmod 11$

   Question: $i = \log_2(3) = \log_2(2^i) = ?$

   Very hard computational problem!

9.3 Attacks for the DL Problem

1. **Brute force**:

   check:

   $\alpha^1 \stackrel{?}{=} \beta$

   $\alpha^2 \stackrel{?}{=} \beta$

   $\vdots$

   $\alpha^i \stackrel{?}{=} \beta$

   Complexity: $\mathcal{O}(|\mathcal{G}|)$ steps.

   Example: DL in $Z_p^* \approx \frac{p-1}{2}$ tests

   minimum security requirement $\Rightarrow p - 1 = |\mathcal{G}| \geq 2^{80}$

2. **Shank's algorithm (Baby-step giant-step) and Pollard's-ρ method**:

   Further reading: p. 165 in [Sti95].

   Complexity: $\mathcal{O}(\sqrt{|\mathcal{G}|})$ steps (for both algorithms).

   Example: DL in $Z_p^* \approx \sqrt{p}$ steps

   minimum security requirement $\Rightarrow p - 1 = |\mathcal{G}| \geq 2^{160}$

3. **Pohlig-Hellman algorithm**:

   Let $|\mathcal{G}| = p_1 \cdot p_2 \cdots \underbrace{p_l}_{largest\ \ prime}$

   Complexity: $\mathcal{O}(\sqrt{p_l})$ steps.

   Example: DL in Z_p^*: p_l of $(p-1)$ must be $\geq 2^{160}$

   minimum security requirement $\Rightarrow p_l \geq 2^{160}$

4. **Index-Calculus method**:

   Further reading: [AM97].

   Applies only to Z_p^* and Galois fields GF(2^k)

   Complexity: $\mathcal{O}\ (e^{(1+\mathcal{O}(1))\sqrt{\ln(p)\ln(\ln(p))}})$ steps.

   Example: DL in Z_p^*: minimum security requirement $\Rightarrow p \geq 2^{1024}$

Remark: Index-Calculus is more powerful against DL in Galois Fields GF(2^k) than against DL in Z_p^*.

9.4 Diffie-Hellman Key Exchange

Remarks:

- Proposed in 1976 in Diffie-Hellman paper.
- Used in many practical protocols.
- Can be based on any DL problem.

9.4.1 Protocol

Set-up:

1. Find a large prime p.
2. Find a primitive element α of Z_p^* or of a subgroup of Z_p^*.

Protocol:

Alice		Bob
pick $k_{prA} = a_A \in \{2, 3, \ldots, p-1\}$		pick $k_{prB} = a_B \in \{2, 3, \ldots, p-1\}$
compute $k_{pubA} = b_A = \alpha^{a_A} \bmod p$		compute $k_{pubB} = b_B = \alpha^{a_B} \bmod p$
	$\xrightarrow{b_A}$	
	$\xleftarrow{b_B}$	
$k_{AB} = b_B^{a_A} = (\alpha^{a_B})^{a_A}$		$k_{AB} = b_A^{a_B} = (\alpha^{a_A})^{a_B}$

Session key $k_{ses} = k_{AB} = \alpha^{a_B \cdot a_A} = \alpha^{a_A \cdot a_B} \bmod p$.

9.4.2 Security

Question: Which information does Oscar have?

Answer: α, p, b_A, b_B.

Diffie-Hellman Problem:

$$\boxed{\text{Given } b_A = \alpha^{a_A} \bmod p, b_B = \alpha^{a_B} \bmod p, \text{ and } \alpha \text{ find } \alpha^{a_A \cdot a_B} \bmod p.}$$

One solution to the D-H problem:

1. Solve DL problem: $a_A = \log_\alpha(b_A) \bmod p$.

2. Compute: $b_B^{a_A} = (\alpha^{a_B})^{a_A} = \alpha^{a_A \cdot a_B} \bmod p$.

   Choose $p \geq 2^{1024}$.

Note:

> There is no proof that the DL problem is the only solution to the D-H problem!
>
> However, it is conjectured.

Chapter 10

Elliptic Curve Cryptosystem

Further Reading:

Chapter 6 in [Kob94].

Book by Alfred Menezes [Men93].

Remarks:

- Relatively new cryptosystem, suggested independently:
  → 1987 by Koblitz at the University of Washington,
  → 1986 by Miller at IBM.
- It is believed to be more secure than RSA/DL in Z_p^*, but uses arithmetic with much shorter numbers ($\approx 160 - 256$ bits vs. $1024 - 2048$ bits).
- It can be used instead of D-H and other DL-based algorithms.

Drawbacks:

- Not as well studied as RSA and DL-base public-key schemes.
- It is conceptually more difficult.
- Finding secure curves in the set-up phase is computationally expensive.

10.1 Elliptic Curves

Goal: To find another instance for the DL problem in cyclic groups.

Question: What is the equation $x^2 + y^2 = r^2$ over reals?

Answer: It is a circle.

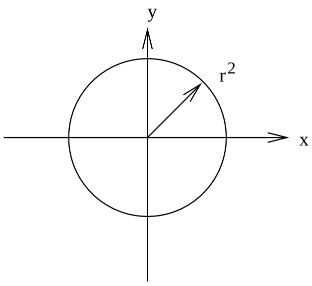


Figure 10.1: $x^2 + y^2 = r^2$ over reals

Question: What is the equation $a \cdot x^2 + b \cdot y^2 = c$ over reals?

Answer: It is an ellipsis.

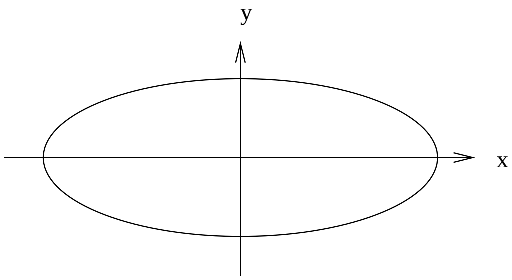


Figure 10.2: $a \cdot x^2 + b \cdot y^2 = c$ over reals

Note:

> There are only certain points (x,y) which fulfill the equation. For example the point $(x = r, y = 1)$ fulfills the equation of a circle.

Definition 10.1.1 *The* elliptic curve *over Z_p, $p > 3$, is a set of all pairs $(x, y) \in Z_p$ which fulfill:*

$$y^2 \equiv x^3 + a \cdot x + b \bmod p$$

where

$$a, b, \in Z_p$$

and

$$4 \cdot a^3 + 27 \cdot b^2 \neq 0 \bmod p$$

Question: How does $y^2 = x^3 + a \cdot x + b$ look over reals?

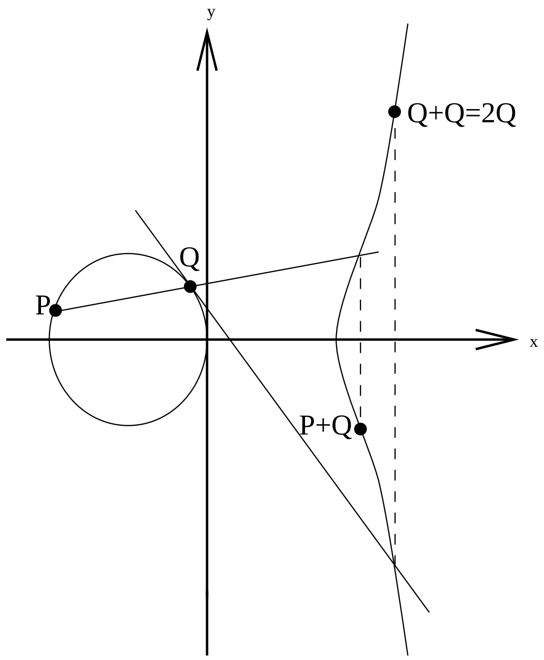


Figure 10.3: $y^2 = x^3 + a \cdot x + b$ over the reals

Goal: Finding a (cyclic) group $(\mathcal{G}, \circ)$ so that we can use the DL problem as a one-way function.

We have a set (points on the curve). We "only" need a group operation on the points.

Group $\mathcal{G}$: Points on the curve given by (x, y).

Operation $\circ$: $P + Q = (x_1, y_1) + (x_2, y_2) = R = (x_3, y_3)$.

Question: How do we find R?

Answer: First geometrically.

a) $P \neq Q \rightarrow$ line through P and Q and mirror point of third interception along the x-axis.

b) $P = Q \Rightarrow P + Q = 2Q \rightarrow$ tangent line through Q and mirror point of second intersection along the x-axis.

Point Addition (group operation):

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \bmod p \\ y_3 &= \lambda(x_1 - x_3) - y_1 \bmod p \end{aligned}$$

where

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \bmod p & ; \text{if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} \bmod p & ; \text{if } P = Q \end{cases}$$

Remarks:

- If $x_1 \equiv x_2 \bmod p$ and $y_1 \equiv -y_2 \bmod p$, then $P + Q = \mathcal{O}$ which is an abstract point at infinity.
- $\mathcal{O}$ is the neutral element of the group: $P + \mathcal{O} = P$; for all P.
- Additive inverse of any point $(x, y) = P$ is $P + (-P) = \mathcal{O}$ such that $(x, y) + (x, -y) = \mathcal{O}$.

Theorem 10.1.1 *The points on an elliptic curve together with $\mathcal{O}$ have cyclic subgroups.*

Remark: Under certain conditions all points on an elliptic curve form a cyclic group as the following example shows.

Example: Finding all points on the curve E: $y^2 \equiv x^3 + x + 6 \bmod 11$.

$\#E = 13$.

primitive element $\rightarrow \alpha = (2,7) \Rightarrow$ generates all points.

$2\alpha = \alpha + \alpha = (2,7) + (2,7) = (x_3, y_3)$

$\lambda = \frac{3x_1^2+a}{2y_1} = (2 \cdot 7)^{-1}(3 \cdot 4 + 1) = 3^{-1} \cdot 13 \equiv 4 \cdot 13 \equiv 4 \cdot 2 = 8 \bmod 11$

$x_3 = \lambda^2 - x_1 - x_2 = 8^2 - 2 - 2 = 60 \equiv 5 \bmod 11$

$y_3 = \lambda(x_1 - x_3) - y_1 = 8(2-5) - 7 = -24 - 7 = -31 \equiv 2 \bmod 11$

$2\alpha = (2,7) + (2,7) = (5,2)$

$3\alpha = 2\alpha + \alpha = \ldots$

$\vdots$

$12\alpha = 11\alpha + \alpha = (2,4)$

$13\alpha = 12\alpha + \alpha = (2,4) + (2,7) = (2,4) + (2,-4) = \mathcal{O}$

$14\alpha = 13\alpha + \alpha = \mathcal{O} + \alpha = \alpha$

$\vdots$

All 12 non-zero elements together with $\mathcal{O}$ form a cyclic group.

$\alpha = (2,7)$	$2\alpha = (5,2)$	$3\alpha = (8,3)$
$4\alpha = (10,2)$	$5\alpha = (3,6)$	$6\alpha = (7,9)$
$7\alpha = (7,2)$	$8\alpha = (3,5)$	$9\alpha = (10,9)$
$10\alpha = (8,8)$	$11\alpha = (5,9)$	$12\alpha = (2,4)$

Table 10.1: Non-zero elements of the group over $y^2 \equiv x^3 + x + 6 \bmod 11$

Remark: In general, finding of the group order $\#E$ is computationally very complex.

10.2 Cryptosystems

10.2.1 Diffie-Hellman Key Exchange

The cryptosystem is completely analogous to D-H in Z_p^*.

Set-up:

1. Choose E: $y^2 \equiv x^3 + a \cdot x + b \bmod p$.
2. Choose primitive element $\alpha = (x_\alpha, y_\alpha)$.

Protocol:

Alice		Bob
choose $k_{prA} = a_A \in \{2, 3, \ldots, \#E - 1\}$		choose $k_{prB} = a_B \in \{2, 3, \ldots, \#E - 1\}$
compute $k_{pubA} = b_A = a_A \cdot \alpha = (x_A, y_A)$		compute $k_{pubB} = b_B = a_B \cdot \alpha = (x_B, y_B)$
	$\xrightarrow{b_A}$	
	$\xleftarrow{b_B}$	
compute $a_A \cdot b_B = a_A \cdot a_B \cdot \alpha = (x_k, y_k)$		compute $a_B \cdot b_A = a_B \cdot a_A \cdot \alpha = (x_k, y_k)$
$k_{AB} = x_k \in Z_p$		$k_{AB} = x_k \in Z_p$

Security:

$$\text{Diffie-Hellman problem for elliptic curves} \begin{cases} \text{Oscar knows:} & E, p, \alpha, b_A = a_A \cdot \alpha, b_B = a_B \cdot \alpha \\ \text{Oscar wants to know:} & k_{AB} = a_A \cdot a_B \cdot \alpha \end{cases}$$

<u>One</u> possible solution to the D-H problem for elliptic curves:

1. Compute *discrete logarithm*:

   Given α and $\underbrace{\alpha + \alpha + \ldots + \alpha}_{a_A \ \ times} = b_A$, find a_A.

2. Compute $a_A \cdot b_B = a_A \cdot a_B \cdot \alpha$.

Attacks:

- Only possible attacks against elliptic curves are the Pohlig-Hellman scheme together with Shank's algorithm or Pollard's-Rho method.
  $\Rightarrow \#E$ must have one large prime factor p_l
  $\Rightarrow 2^{160} \leq p_l \leq 2^{250}$.

- So-called "Koblitz curves" (curves with $a, b \in \{0, 1\}$)

- For *supersingular* elliptic curves over GF(2^n), DL in elliptic curves can be solved by solving DL in GF($2^{k \cdot n}$); $k \leq 6$.
  $\Rightarrow$ stay away from supersingular curves despite of possible faster implementations.

- Powerful index-calculus method attacks are not applicable (as of yet).

10.2.2 Menezes-Vanstone Encryption

Set-up:

1. Choose E: $y^2 \equiv x^3 + a \cdot x + b \bmod p$.
2. Choose primitive element $\alpha = (x_\alpha, y_\alpha)$.
3. Pick random integer $a \in \{2, 3, \ldots, \#E - 1\}$.
4. Compute $a \cdot \alpha = \beta = (x_\beta, y_\beta)$.
5. Public Key: $k_{pub} = (E, p, \alpha, \beta)$.
6. Private Key: $k_{pr} = (a)$.

Encryption:

1. Pick random $k \in \{2, 3, \ldots, \#E - 1\}$. Compute $k \cdot \beta = (c_1, c_2)$.

2. Encrypt $e_{k_{pub}}(x, k) = (Y_0, Y_1, Y_2)$.

   $Y_0 = k \cdot \alpha \rightarrow$ point on the elliptic curve.

   $Y_1 = c_1 \cdot x_1 \bmod p \rightarrow$ integer.

   $Y_2 = c_2 \cdot x_2 \bmod p \rightarrow$ integer.

Decryption:

1. Compute $a \cdot Y_0 = (c_1, c_2)$.

   $a \cdot Y_0 = a \cdot k \cdot \alpha = k \cdot \beta = (c_1, c_2)$.

2. Decrypt: $d_{k_{pr}}(Y_0, Y_1, Y_2) = (Y_1 \cdot c_1^{-1} \bmod p, Y_2 \cdot c_2^{-1} \bmod p) = (x_1, x_2)$.

Remark: The disadvantage of this scheme is the message expansion factor:

$$\frac{\# \ \ bits \ \ y}{\# \ \ bits \ \ x} = \frac{4\lceil \log_2 p \rceil}{2\lceil \log_2 p \rceil} = 2$$

10.3 Implementation

1. Hardware:

   - Approximatly 0.2 msec for an elliptic curve point multiplication with 167 bits on an FPGA [OP00].

2. Software:

   - One elliptic curve point multiplication $a \cdot P$ in less than 10 msec over GF(2^{155}).

   - Implementation on 8-bit smart card processor without coprocessor available

Chapter 11

ElGamal Encryption Scheme

11.1 Cryptosystem

Remarks:

- Published in 1985.
- Based on the DL problem in Z_p^* or $\mathrm{GF}(2^k)$.
- Extension of the D-H key exchange for encryption.

Protocol:

Alice		Bob
choose private key $k_{prA} = a_A$		choose private key $k_{prB} = a_B$
compute $k_{pubA} = \alpha^{a_A} \bmod p = b_A$		compute $k_{pubB} = \alpha^{a_B} \bmod p = b_B$
	$\xrightarrow{b_A}$	
	$\xleftarrow{b_B}$	
$k_{AB} = b_B^{a_A} = \alpha^{a_A a_B} \bmod p$		$k_{AB} = b_A^{a_B} = \alpha^{a_B a_A} \bmod p$
$y = x \cdot k_{AB} \bmod p$	$\xrightarrow{y}$	
		$x = y \cdot k_{AB}^{-1} \bmod p$

ElGamal:

Set-up:

1. Choose large prime p.
2. Choose primitive element $\alpha \in Z_p^*$.
3. Choose secret key $a \in \{2, 3, \ldots, p-2\}$.
4. Compute $\beta = \alpha^a \bmod p$.
5. Public Key: $K_{pub} = (p, \alpha, \beta)$.
6. Private Key: $K_{pr} = (a)$.

Encryption:

1. Choose $k \in \{2, 3, \ldots, p-2\}$.
2. $Y_1 = \alpha^k \bmod p$.
3. $Y_2 = x \cdot \beta^k \bmod p$.
4. Encryption: $= e_{k_{pub}}(x, k) = (Y_1, Y_2)$.

Decryption:

$x = d_{k_{pr}}(Y_1, Y_2) = Y_2(Y_1^a)^{-1} \bmod p.$

Question: How does the ElGamal scheme work?

$$\begin{aligned}
d_{k_{pr}}(Y_1, Y_2) &= Y_2(Y_1^a)^{-1} \\
&= x \cdot \beta^k((\alpha^k)^a)^{-1} \rightarrow \text{ but } \beta = \alpha^a \\
&= x(\alpha^a)^k((\alpha^k)^a)^{-1} \\
&= x \cdot \alpha^{ak} \cdot \alpha^{-ak} \\
&= x
\end{aligned}$$

Remarks:

- ElGamal is essentially an extension of the D-H key exchange protocol.
- $\left.\begin{array}{l} Y_2 = x_1 \cdot \beta^k \\ Y_3 = x_2 \cdot \beta^k \end{array}\right\}$ if x_1 is known, β^k can be found from Y_2.
  Thus for every message block x_i choose a new k!
- Message expansion factor
$$\frac{\#\ \ of\ \ y\ \ bits}{\#\ \ of\ \ x\ \ bits} = \frac{2\lceil \log 2p_y \rceil}{\lceil \log 2p_x \rceil} = 2$$
.

11.2 Computational Aspects

11.2.1 Encryption

$$\left.\begin{array}{l} Y_1 = \alpha^k \bmod p \\ Y_2 = x \cdot \beta^k \bmod p \end{array}\right\} \text{apply the square-and-multiply for exponentiation}$$

11.2.2 Decryption

$x = d_{k_{pr}}(Y_1, Y_2) = Y_2 (Y_1^a)^{-1} \bmod p$.

Question: How can $(Y_1^a)^{-1}$ be computed efficiently?

Derivation: $b \in Z_p^*$:

$$\begin{aligned} b^e = b^{q(p-1)+r} &= (b^{p-1})^q \cdot b^r \\ &= 1^q \cdot b^r \bmod p \\ &= b^r \bmod p \\ \Rightarrow e &= r \bmod (p-1) \end{aligned}$$

$$\boxed{\text{Thus, } b^e \equiv b^{e \bmod (p-1)} \bmod p, \text{ where } b \in Z_p^* \text{ and } e \in Z}$$

The above derivation can be used for decryption:

$$\begin{aligned}(Y_1^a)^{-1} = Y_1^{-a} &= Y_1^{-a \bmod (p-1)} \bmod p \\ &= Y_1^{p-1-a} \bmod p\end{aligned}$$

Note: $Y_1^{p-1-a} \bmod p$ can be computed using the square-and-multiply algorithm.

11.3 Security of ElGamal

Oscar knows: $p, \alpha, \beta = \alpha^a, Y_1 = \alpha^k, Y_2 = x \cdot \beta^k$.

Oscar wants to know: x

- He attempts to find the secret key a:

  1. $a = \log_\alpha \beta \bmod p \leftarrow$ hard, DL problem.
  2. $x = Y_2(Y_1^a)^{-1} \bmod p \leftarrow$ easy.

- He attempts to find the random exponent k:

  1. $k = \log_\alpha Y_1 \bmod p \leftarrow$ hard, DL problem.
  2. $Y_2 \cdot \beta^{-k} = x \leftarrow$ easy.

- In both cases Oscar has to compute the DL problem in finite fields (Z_p^* or $GF(2^k)$). He can use index-calculus method which forces us to implement schemes with at least 1024 bits.

Chapter 12

Digital Signatures

Protocols use:

- Private-key algorithms.
- Public-key algorithms.
- Digital Signatures.
- Hash functions.
- Message Authentication Codes.

as building blocks. In practice, protocols are often the most vulnerable part of a cryptosystem. The next two chapters deal with digital signature, message authentication codes (MACs), and hash functions.

12.1 Principle

The idea is similar to a conventional signature where a given message x gets a unique digital signature which is a function of the message and is attached to the message.

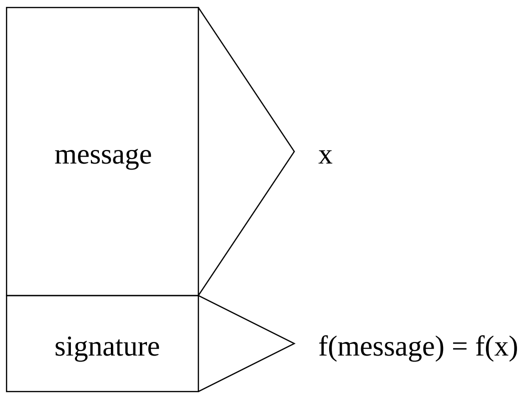


Figure 12.1: Digital signature and message block

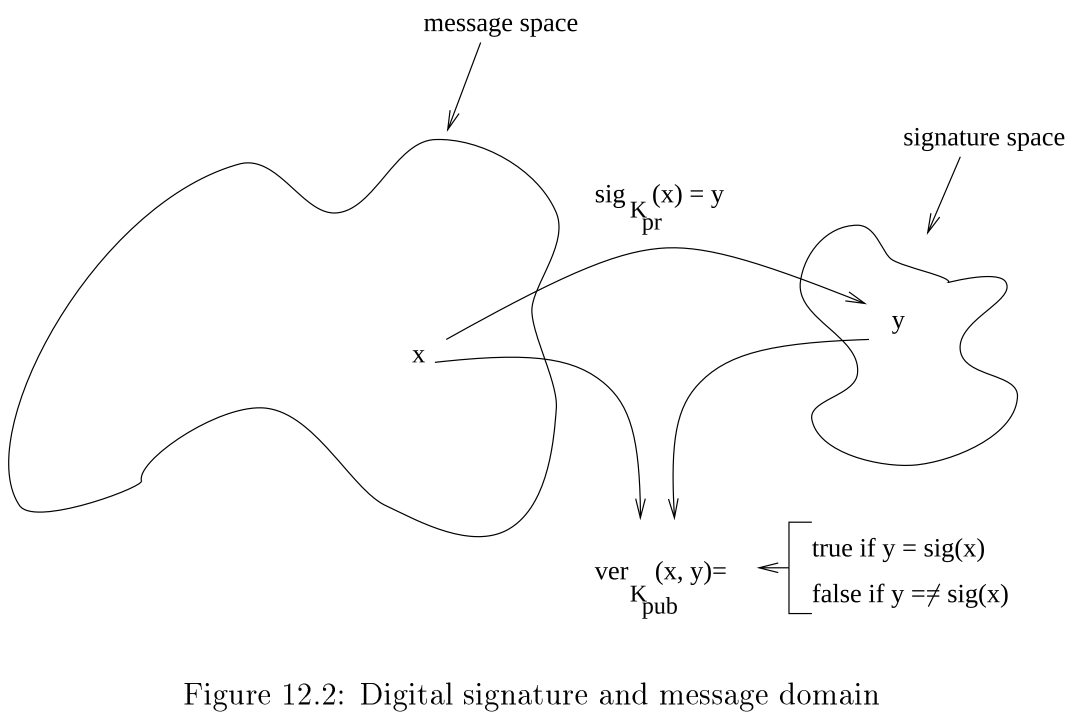


Figure 12.2: Digital signature and message domain

Basic protocol:

1. Bob signs his message x with <u>his</u> private key k_{pr}:

   $\Rightarrow y = sig_{k_{pr}}(x)$.

2. Bob sends (y, x) to Alice.

3. Alice runs the verification function $ver_{k_{pub}}(x, y)$ with <u>Bob's</u> public key.

Properties of digital signatures:

- Only Bob can sign his document (with k_{pr}).
- Everyone can verify the signature (with k_{pub}).
- Authentication: Alice is sure that Bob signed the message.
- Integrity: Message x cannot be altered since that would be detected through verification.
- Non-repudiation

12.2 RSA Signature Scheme

Set-up: $k_{pr} = (p, q, a)$; $k_{pub} = (n, b)$.

General Protocol:

1. Bob computes: $y = sig_{k_{pr}}(x) = e_{k_{pr}}(x) = x^a \bmod n$.

2. Bob sends (x, y) to Alice.

3. Alice verifies:

$$ver_{k_{pub}}(x, y) = d_{k_{pub}}(y) = y^b \begin{cases} = x & \Rightarrow \text{ true} \\ \neq x & \Rightarrow \text{ false} \end{cases}$$

Question: Why does it work?

$$d_{k_{pub}}(y) = d_{k_{pub}}(e_{k_{pr}}(x)) = x.$$

Remark:

- The role of public/private key are exchanged if compared with RSA public-key encryption.
- This algorithm was standardized in ISO/IEC 9796.

Drawback:

Oscar can generate a valid signature for a *random* message x:

1. Choose signature $y \in Z_n$.
2. Encrypt: $x = e_{k_{pub}}(y) = y^b \mod n \rightarrow$ outcome x **cannot** be controlled.
3. Send (x, y) to Alice.
4. Alice verifies: $ver_{k_{pub}}(x, y)$: $y^b \equiv x \mod n \Rightarrow$ true.

12.3 ElGamal Signature Scheme

Remarks:

- ElGamal signature scheme is different from ElGamal encryption.
- Digital Signature Algorithm (DSA) is a modification of ElGamal signature scheme.
- This scheme was published in 1985.

Set-up:

1. Choose a prime p.
2. Choose primitive element $\alpha \in Z_p^*$.
3. Choose random $a \in \{2, 3, \ldots, p-2\}$.
4. Compute $\beta = \alpha^a \bmod p$.

   Public key: $k_{pub} = (p, \alpha, \beta)$.

   Private key: $k_{pr} = (a)$.

Signing:

1. Choose random $k \in \{0, 1, 2, \ldots, p-2\}$; such that $\gcd(k, p-1) = 1$.
2. Compute signature:

$$\begin{aligned} sig_{k_{pr}}(x, k) &= (\gamma, \delta), \quad \text{where} \\ \gamma &= \alpha^k \bmod p \\ \delta &= (x - a \cdot \gamma)k^{-1} \bmod p-1 \end{aligned}$$

Public verification:

$$ver_{k_{pub}}(x, (\gamma, \delta)) = \beta^\gamma \cdot \gamma^\delta \begin{cases} = \alpha^x \bmod p & \text{valid signature} \\ \neq \alpha^x \bmod p & \text{invalid signature} \end{cases}$$

Question: Why does this scheme work?

$$\begin{aligned} \beta^\gamma \cdot \gamma^\delta &= (\alpha^a)^\gamma (\alpha^k)^{(x-a\cdot\gamma)k^{-1} \bmod (p-1)} \bmod p \\ &= \alpha^{a\cdot\gamma} \cdot \alpha^{k \cdot k^{-1}(x - a\cdot\gamma)} \bmod p \\ &= \alpha^{a\cdot\gamma - a\cdot\gamma + x} = \alpha^x \end{aligned}$$

Chapter 13

Hash Functions

13.1 Introduction

The problem with digital signatures is that long messages require very long signatures. We would like for performance as well as for security reasons to have **one** signature for a message of arbitrary length. The solution to this problem are Hash functions.

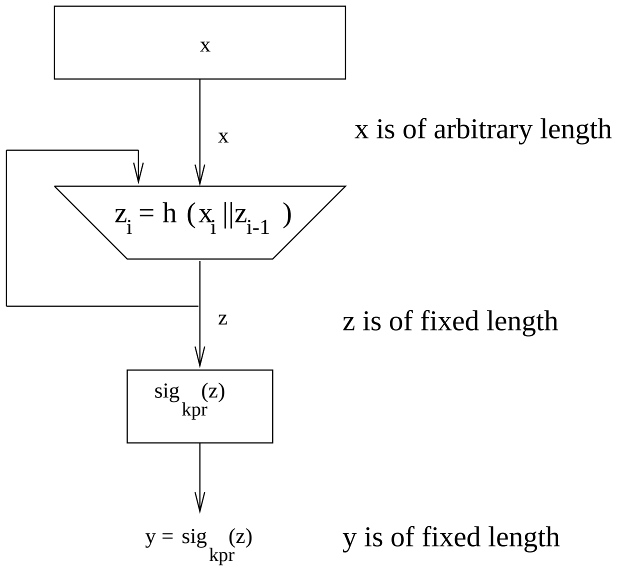


Figure 13.1: Hash functions and digital signatures

Remarks:

- z, x don't have the same length.
- $h(x)$ has no key.
- $h(x)$ is public.

Basic Protocol:

Alice		Bob
		1) $z = h(x)$
		2) $y = sig_{k_{pr}}(z)$
	3) $\xleftarrow{(x,y)}$	
4) $z = h(x)$		
5) $ver_{k_{pub}}(z, y)$		

Potential hash function properties

a) *One-way:* for (almost) all given output z, it is impossible to find any input x such that $h(x) = z$.

b) *Weak collision resistant:* given x, and thus $h(x)$, it is impossible to find any x' such that $h(x) = h(x')$.

c) *Strong collision resistant:* it is impossible to find any two pairs x, x' such that $h(x) = h(x')$.

Requirements for a hash function (Adopted from [Sta95])

1. $h(x)$ can be applied to x of any size.
2. $h(x)$ produces a fixed length output.
3. $h(x)$ is relatively easy to compute in software and hardware.
4. $h(x)$ is one-way.
5. $h(x)$ is weak collision resistant.
6. $h(x)$ is strong collision resistant.

Discussion:

- (1) — (3) are practical requirements
- (4) if $h(x)$ is not one-way, Oscar can compute x from $h(x)$ in cases where x is encrypted.
- (5) if $h(x)$ is not weak collission free, Oscar can replace x with x'.

Alice		Oscar		Bob
				$z = h(x)$
			$\xleftarrow{(x,y)}$	$y = sig_{K_{pr}}(z)$
	$\xleftarrow{(y,x')}$			
$z = h(x') = h(x)$				
$ver_{K_{pub}}(z, y) = true$				

- (6) if $h(x)$ is not strong collission free, Oscar runs the following attack:

  a) Choose legitimate message x_1 and fraudulent message x_2

b) Alter x_1 and x_2 at "non-visible" location, i.e. replace tabs through spaces, append returns, etc., until $h(x'_1) = h(x'_2)$ (*Note:* e.g. 64 alteration locations allow 2^{64} versions of a message with 2^{64} different hash values).

c) Let Bob sign $x'_1 \rightarrow (x'_1, sig_{K_{pr}}(h(x'_1))$

d) Replace $x'_1 \rightarrow x'_2$ and $(x'_2, sig_{K_{pr}}(h(x'_2))$

13.2 Security Considerations

Question: How many people are needed at a party so that there is a 50% chance that at least two people have the same birthday?

In general, given a large set with n different values:

$$P(\text{no collission among } k \text{ random elements}) = \underbrace{\underbrace{\underbrace{\left(1-\frac{1}{n}\right)}_{k=2 \text{ elt.}} \left(1-\frac{2}{n}\right)}_{k=3 \text{ elt.}} \cdots \left(1-\frac{k-1}{n}\right)}_{k \text{ elt.}} = \prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)$$

Often n is large ($n = 365$ in birthday paradox, $n = 2^{160}$ in hash functions).
Recall:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$$

if $x << 1$

$$e^{-x} \approx 1 - x$$

Thus,

$$P(\text{no collision}) \approx \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\frac{1}{n}} e^{-\frac{2}{n}} e^{-\frac{3}{n}} \cdots e^{-\frac{k-1}{n}}$$

$$\prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\frac{1+2+3+\cdots+k-1}{n}}$$

Rewriting the exponent with the help of the following identity:

$$1 + 2 + 3 + \cdots + k - 1 = k(k-1)/2$$

We obtain,

$$P(\text{no collission}) \approx e^{-\frac{k(k-1)}{2n}}$$

Define ϵ as

$$\begin{aligned}
P(\text{at least one collission}) &\stackrel{DEF}{=} \epsilon \;\approx\; 1 - e^{-\frac{k(k-1)}{2n}} \\
1 - \epsilon &\approx e^{-\frac{k(k-1)}{2n}} \\
\ln(1-\epsilon) &\approx -\frac{k(k-1)}{2n} \\
k(k+1) &\approx -2n\ln(1-\epsilon) \;=\; 2n\ln\left(\frac{1}{1-\epsilon}\right)
\end{aligned}$$

If $k >> 1$, then

$$\begin{aligned}
k^2 &\approx k(k-1) \approx 2n\ln\left(\frac{1}{1-\epsilon}\right) \\
k &\approx \sqrt{2n\ln\left(\frac{1}{1-\epsilon}\right)}
\end{aligned}$$

Example:

$$k(\epsilon = 0.5) \approx \sqrt{2n\ln\left(\frac{1}{1-0.5}\right)} = \sqrt{2\ln 2}\sqrt{n} = 1.18\sqrt{n}$$

$\Rightarrow$ A collission in a set of n values is found after about $\sqrt{n}$ trials with a probability of 0.5. In other words, hash funtion with 40 bit output $\Rightarrow$ collission after $\approx \sqrt{2^{40}} = 2^{20}$ trials.

$\Rightarrow$ In order to provide collision resistance in practice, the output space of the hash function should contain at least 2^{160} elements, that is, the hash function should have at least 160 output bits. Finding a collision takes then roughly $\sqrt{2^{160}} = 2^{80}$ steps.

13.3 Hash Algorithms

Overview:

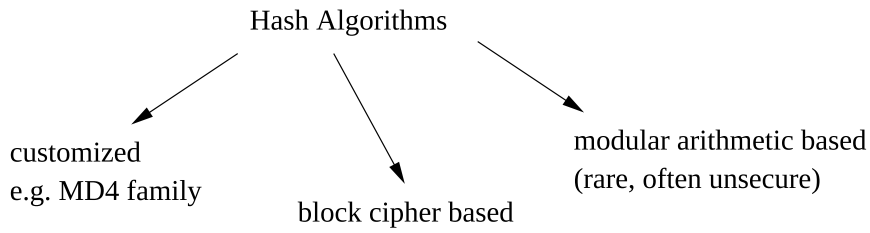


Figure 13.2: Family of Hash Algorithms

a) MD4–family

1. **SHA-1**

Output: 160 bits ⇒ input size for DSS.
Input: 512 bit chunks of message x.
Operations: bitwise AND, OR, XOR, complement and cyclic shift.

2. **RIPE-MD 160**

Output: 160 bits.
Input: 512 bit chunks of message x.
Operations: same as SHA but runs two algorithms in parallel whose outputs are combined after each round.

b) Hash functions from block ciphers

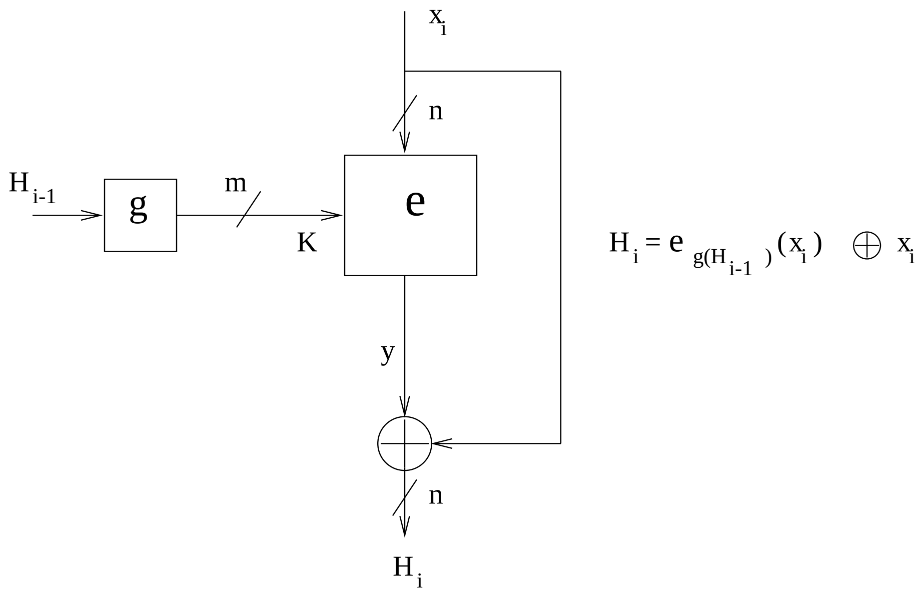


Figure 13.3: Hash Functions from Block Ciphers

where g is a simple n-to-m bit mapping function (if $n = m$, g can be the identity mapping)

Last output H_l is the hash of the whole message x_1,x_2,. . .,x_l

Also secure are:

- $H_i = H_{i-1} \oplus e_{x_i}(H_{i-1})$
- $H_i = H_{i-1} \oplus x_i \oplus e_{g(H_{i-1})}(x_i)$

Remark:

For block ciphers with less than 128 bit block length, different techniques must be used (Sec. 9.4.1 (ii) in [AM97])

Chapter 14

Message Authentication Codes (MACs)

Other names: “cryptographic checksum” or “keyed hash function”.

Private-key based.

14.1 Principle

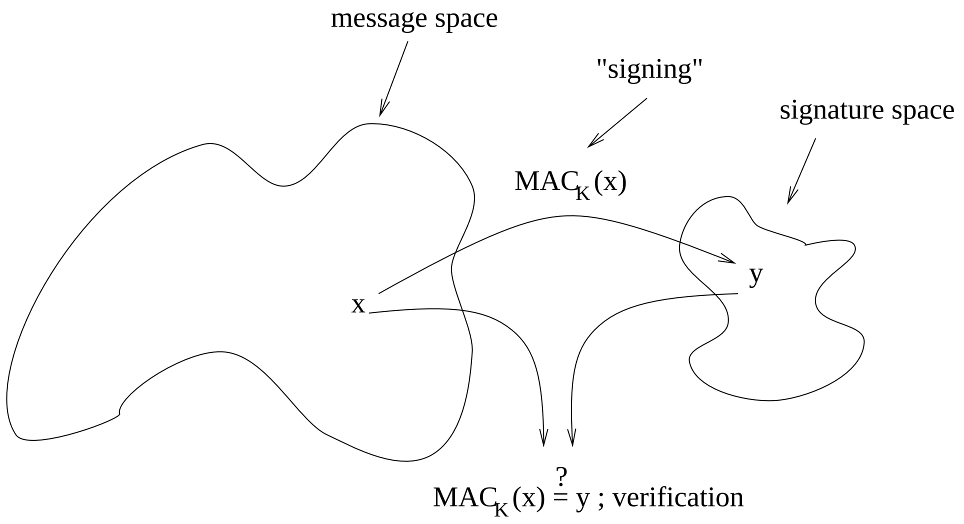


Figure 14.1: MAC and message domain

Protocol:

Alice		Bob
		1) $y = \mathrm{MAC}_K(x)$
	2) $\xleftarrow{(x,y)}$	
3) $y' = \mathrm{MAC}_K(x)$ $y' \stackrel{?}{=} y$		

Properties:

1. Generate signature for a given message.
2. Private-key based: signing and verifying party must share a secret key.
3. Accepts messages of arbitrary length and generates fixed size signature.

Properties 2 and 3 are different from digital signatures.

Idea: To use block-cipher's one of the chaining modes to generate signature.

14.2 MACs from Block Ciphers

CBC mode:

$y_0 = e_k(x_0 \oplus IV) = e_k(x_0 \oplus 0000\ldots)$

$y_i = e_k(x_i \oplus y_{i-1})$

$X = x_0, x_1, \ldots, x_{m-1}$

$MAC_k(x) = y_{m-1}$

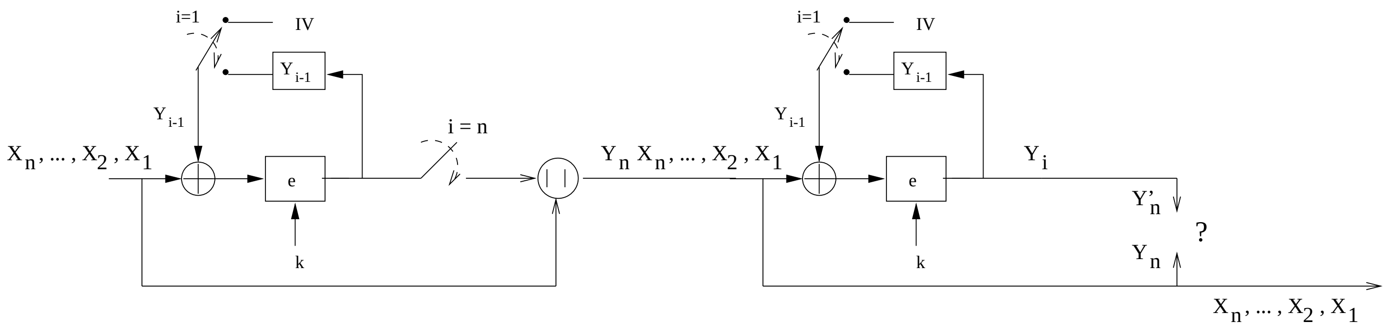


Figure 14.2: MAC in a CBC mode

Verification: Run the same process on the receiving end.

Remark: CBC with DES is standardized (ANSI X9.17).

14.3 HMAC

- Popular in modern protocols such as SSL.
- Attractive property: HMAC can be *proven* to be secure under certain assumptions about the hash function. "Secure" means here that the hash function has to be broken in order to break the HMAC.
- **Basic idea:** Hash a secret key K together with the message M and consider the hash output the authentication tag for the message: $H(K||M)$.
- **Details:**

  $\text{HMAC}_K(M) = H\left[(K^+ \oplus \text{opad})||H\left[(K^+ \oplus \text{ipad})||M\right]\right]$

  where

  $K^+ = K$ padded with zeros on the left so that the result is b bits in length (where b is the number of bits in a block).

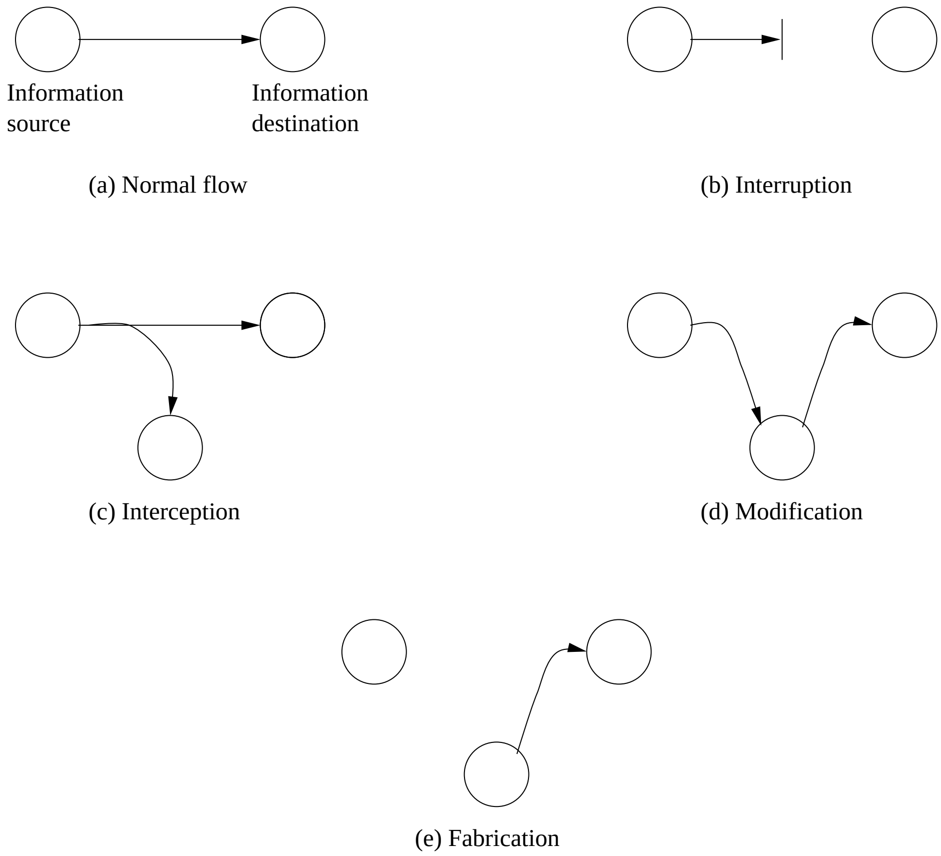
ipad = 00110110 repeated $b/8$ times.

opad = 01011010 repeated $b/8$ times.

Chapter 15

Security Services

15.1 Attacks Against Information Systems



Remarks:

- Passive attacks: (c) $\rightarrow$ interception.
- Active attacks: (b) $\rightarrow$ interruption, (d) $\rightarrow$ modification, (e) $\rightarrow$ fabrication.

15.2 Introduction

Security Services are **goals** which information security systems try to achieve. Note that cryptography is only **one module** in information security systems.

The main security services are:

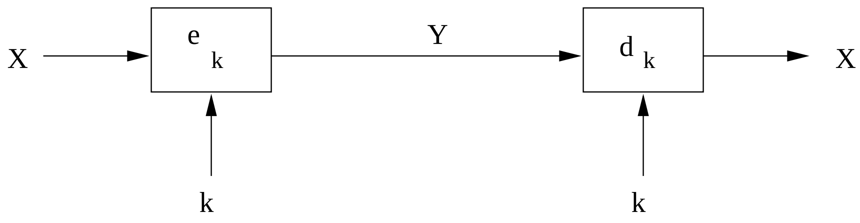
- *Confidentiality/Privacy.* Information is kept secret from all but authorized parties.
- *(Message/Sender) Authentication.* Ensures that the sender of a message is who she/he claims to be.
- *Integrity.* Ensures that a message has not been modified in transit.
- *Non-repudiation.* Ensures that the sender of a message can not deny the creation of the message.
- *Identification/Entity Authentication.* Establishing of the identity of an entity (e.g. a person, computer, credit card).
- *Access Control.* Restricting access to the resources to privileged entitites.

Remark: Message Authentication implies data integrity; the opposite is not true.

15.3 Privacy

Tool: Encryption algorithm.

a) Private-Key



Provides:

−privacy

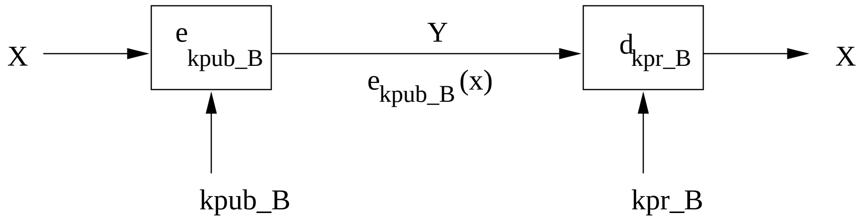
−message authentication and thus
−integrity
−no non-repudiation

only if Bob can distinguish between valid and invalid X and if there are only two parties.

Remark:

In practice, authentication and integrity are often achieved with MACs (Chapter 14)

b) Public-Key



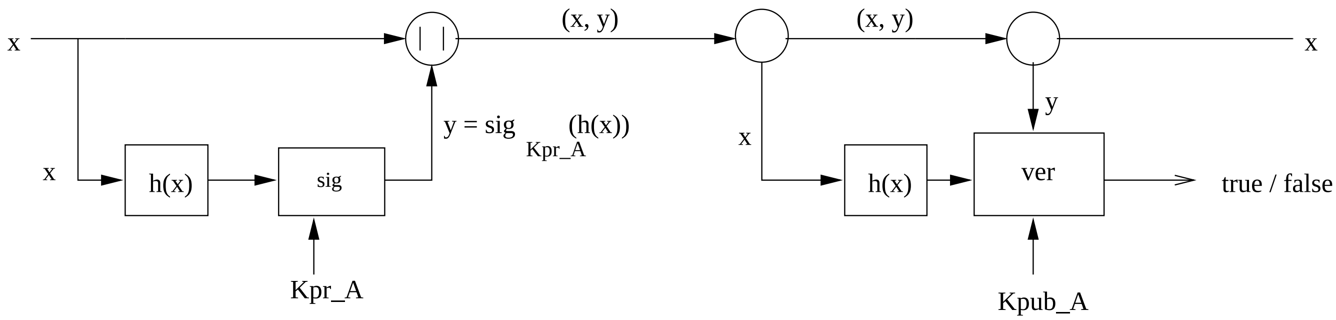
Provides:

- privacy
- integrity (if invalid x can e detected)
- no message authentication

15.4 Integrity and Sender Authentication

Recall: Sender authentication implies integrity.

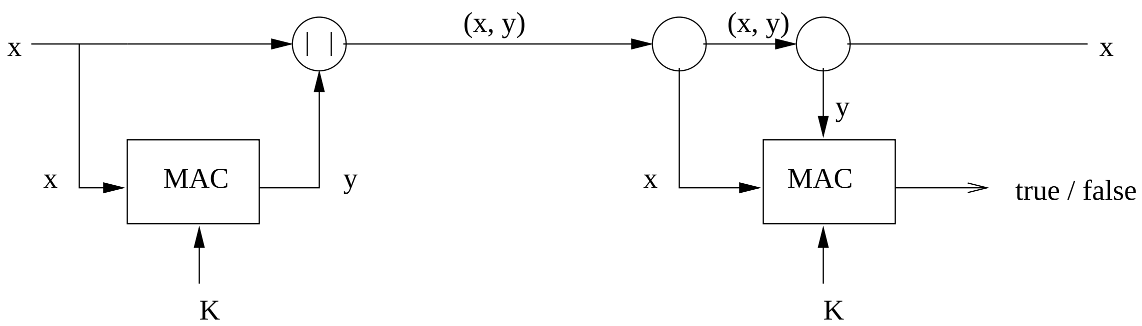
15.4.1 Digital Signatures



Provides:

- integrity
- sender authentication
- non-repudiation (only Alice can construct valid signature)

15.4.2 MACs



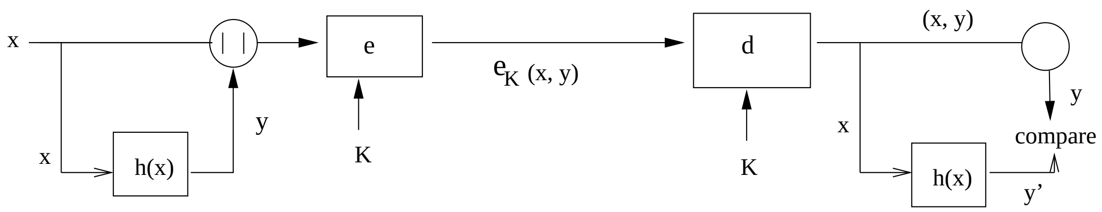
Provides:

- integrity

- authentication

- no non-repudiation

15.4.3 Integrity and Encryption



Provides:

- privacy

- integrity

- authentication

- no non-repudiation

Remark:

- Instead of hash functions, MACs are also possible. In this case: $c = e_{K_1}(x, \text{MAC}_{K_2}(y))$.

- This scheme adds strong authentication and integrity to an encryption-protocol with very little computational overhead.

Chapter 16

Key Establishment

16.1 Introduction

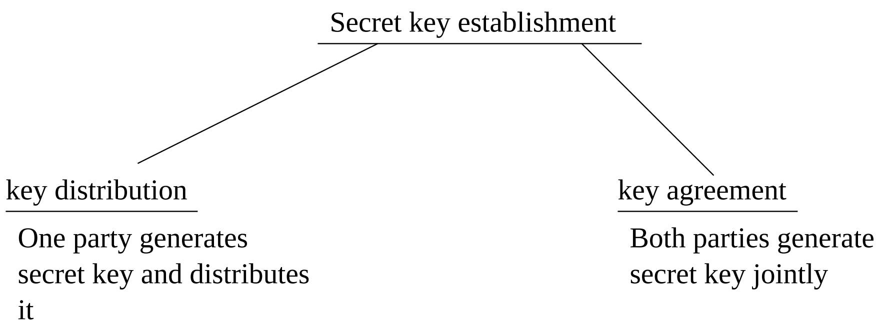


Figure 16.1: Key establishment schemes

Remark:

Some schemes make use of trusted authority (TA) which is trusted by and can communicate with all users.

16.2 Private-Key Approaches

16.2.1 The n^2 Key Distribution Problem

TA generates a key for every pair of users:

Example: $n = 4$ users.

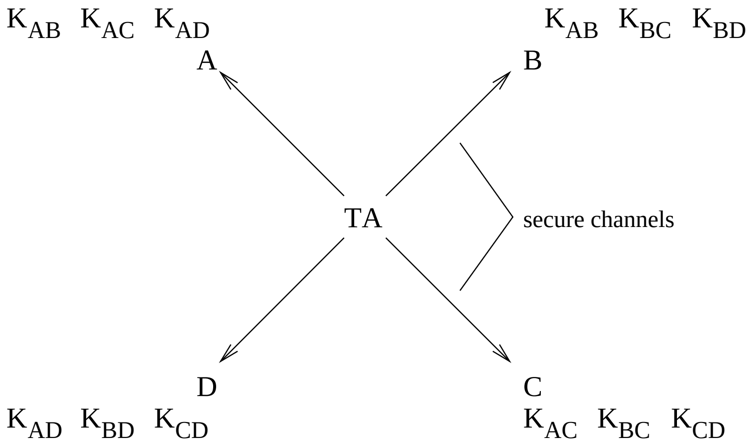


Figure 16.2: The role of the Trusted Authority

Drawbacks:

- n secure channels are needed
- each user must store $n-1$ keys
- TA must transmit $n(n-1)$ keys
- TA must generate $\frac{n(n-1)}{2} \approx \frac{n^2}{2}$ keys
- every new network user makes updates at all other user as of necessary $\Rightarrow$ scales badly

16.2.2 Key Distribution Center (KDC)

TA is a KDC: TA shares secret key with each user and generates session keys.

a) Basic protocol:

- k_s = session key between Alice and Bob
- $k_{A,KDC}$ = secret key between Alice and KDC (Key encryption key, KEK)
- $k_{B,KDC}$ = secret key between Bob and KDC (Key encryption key, KEK)

Alice		KDC		Bob
	$\xleftarrow{e_{k_{A,KDC}}(k_s)=y_A}$		$\xrightarrow{e_{k_{B,KDC}}(k_s)=y_B}$	
$k_s = d_{k_{A,KDC}}(y_A)$				$k_s = d_{k_{B,KDC}}(y_B)$
$y = e_{k_s}(x)$		$\xrightarrow{y}$		$x = d_{k_s}(y)$

Remarks:

- TA stores only n keys
- each user U stores only one key

b) Modified (advanced) protocol:

Alice		KDC		Bob
		1a) $y_A = e_{k_A}(k_s)$		
		1b) $y_B = e_{k_B}(k_s)$		
	2) $\xleftarrow{(y_A, y_B)}$			
3) $k_s = d_{k_A}(y_A)$				
4) $y = e_{k_s}(x)$		5) $\xrightarrow{(y, y_B)}$		7) $k_s = d_{k_B}(y_B)$
				6) $x = d_{k_s}(y)$

Remark: This approach is the basis for Kerberos.

16.3 Public-Key Approaches

16.3.1 Man-In-The-Middle Attack

D-H key exchange revised

Set-up:

- find large prime p
- find primitive element $\alpha \in Z_p$

Protocol:

Alice		Bob
pick $k_{prA} = a_A \in \{2, 3, \ldots, p-2\}$		pick $k_{prB} = a_B \in \{2, 3, \ldots, p-2\}$
compute $k_{pubA} = b_A = \alpha^{a_A} \bmod p$		compute $k_{pubB} = b_B = \alpha^{a_B} \bmod p$
	$\xrightarrow{b_A}$	
	$\xleftarrow{b_B}$	
$k_{AB} = b_B^{a_A} = \alpha^{a_A a_B} \bmod p$		$k_{AB} = b_A^{a_B} = \alpha^{a_A a_B} \bmod p$

Security:

1. passive attacks

   $\Rightarrow$ security relies on Diffie-Hellman problem thus $p > 2^{1000}$.

2. active attack

   $\Rightarrow$ *Man-in-the-middle attack:*

Alice		Oscar		Bob
	$\xrightarrow{\alpha^a}$		$\xrightarrow{\alpha^o}$	
	$\xleftarrow{\alpha^o}$		$\xleftarrow{\alpha^b}$	
$k_{AO} = (\alpha^o)^a = \alpha^{ao}$		$k_{AO} = (\alpha^a)^o$		$k_{BO} = (\alpha^o)^b = \alpha^{bo}$
		$k_{BO} = (\alpha^b)^o$		
$y' = e_{k_{AO}}(x)$	$\xrightarrow{y'}$	$x = d_{k_{AO}}(y')$		
		$y'' = e_{k_{BO}}(x)$	$\xrightarrow{y''}$	$x = d_{k_{BO}}(y'')$

Remarks:

- Oscar can read and alter x without detection.
- Underlying Problem: *public keys are not authenticated.*
- **Man-in-the-middle attack applies to all Public-key schemes.**

16.3.2 Certificates

Certificates bind ID information (e.g., name, social security number) to a public key through digital signatures.

General structure of certificates:

1. Each user U:
   - ID(U) = ID information such as user name, e-mail address, SS#, etc.
   - private key: K_{prU}
   - public key: K_{pubU}
2. Certifying Authority (CA):
   - secret signature algorithm sig_{TA}
   - public verification algorithm ver_{TA}
   - certificates for each user U:
     $C(U) = (ID(U), K_{prU}, sig_{TA}(ID(U), K_{prU}))$

General requirement: all users have the correct verification algorithm ver_{TA} with TA's public key.

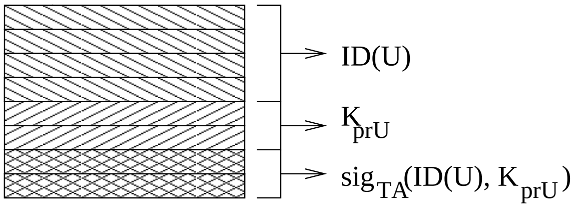


Figure 16.3: General structure of the certificate C(U)

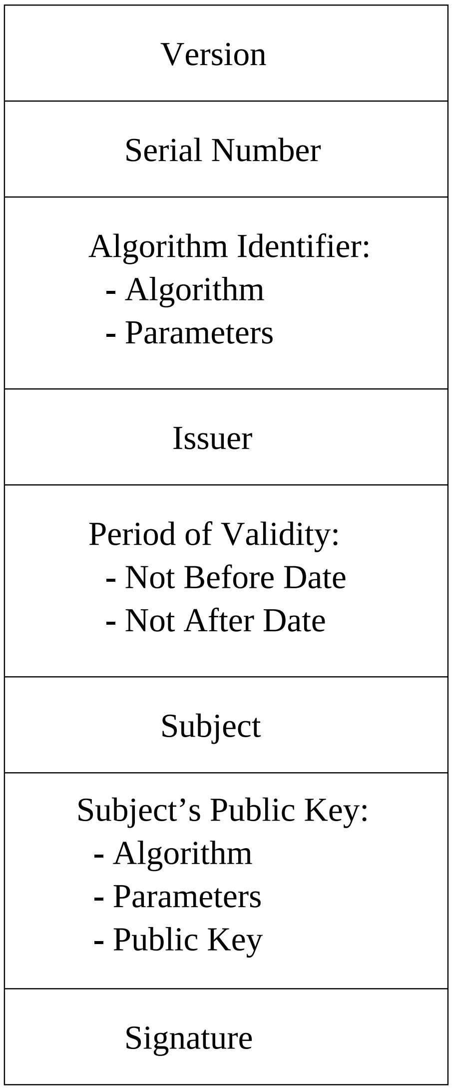


Figure 16.4: Detailed structure of an X.509 certificate

Remarks:

- Certificate structures are specified in X.509, authentication services for the X.500 directory recommendation (CCITT).

16.3.3 Diffie-Hellman Exchange with Certificates

Idea: As standard D-H, but each users's public key is authenticated by a certificate.

Alice		Bob
$K_{pubA} = b_A$		$K_{pubB} = b_B$
$K_{prA} = a_A$		$K_{prB} = a_B$
	$\xleftarrow{C(B)=(ID(B),b_B,sig_{CA}(ID(B),b_B))}$	
	$\xrightarrow{C(A)=(ID(A),b_A,sig_{CA}(ID(A),b_A))}$	
1.) $ver_{CA}(ID(B), b_B)$		1.) $ver_{CA}(ID(A), b_A)$
2.) $k_{AB} = b_B^{a_A} = \alpha^{a_B a_A} = \alpha^{a_A a_B}$		2.) $k_{AB} = b_A^{a_B} = \alpha^{a_A a_B}$

Remaining major problems with CAs:

1. The CA's public key must initially be distributed in an authenticated manner!
2. Identity of user must be established by CA.
3. Certificate Revocation Lists (CRLs) must be distributed.

16.3.4 Authenticated Key Agreement

Idea: Alice and Bob sign their own public keys. Signatures can be correctly verified through certificates.

Set-up:

- public verification key for ver_{TA}
- public prime p
- public primitive element $\alpha \in Z_p$

Protocol:

Alice	TA	Bob
	$\xleftarrow{C(A)=(ID(A),ver_A,sig_{TA}(ID(A),ver_A))}$	
	$\xrightarrow{C(B)=(ID(B),ver_B,sig_{TA}(ID(B),ver_B))}$	
1.) $k_{prA} = a_A$		
2.) $k_{pubA} = b_A = \alpha^{a_A} \bmod p$	$\xrightarrow{b_A}$	
		3.) $k_{prB} = a_B$
		4.) $k_{pubB} = b_B = \alpha^{a_B} \bmod p$
		5.) $k_{AB} = b_A^{a_B} = \alpha^{a_A a_B} \bmod p$
	$\xleftarrow{(C(B),b_B,y_B)}$	6.) $y_B = sig_B(b_B, b_A)$
7.) $ver_{TA}(C(B))$: true/false		
8.) $ver_B(y_B)$: true/false		
9.) $k_{AB} = b_B^{a_A} = \alpha^{a_A a_B} \bmod p$		
10.) $y_A = sig_A(b_A, b_B)$	$\xrightarrow{(C(A),y_A)}$	
		11.) $ver_{TA}(C(A))$: true/false
		12.) $ver_A(y_A)$: true/false

Remark:

This scheme is also known as station-to-station protocol and is the basis for ISO 9798-3.

Chapter 17

Case Study: The Secure Socket Layer (SSL) Protocol

Note:

> This chapter describes the most important security mechanisms of the SSL Protocol. For more details references [Sta99] and Netscape's SSL web page are recommended.

17.1 Introduction

- SSL was developed by Netscape.
- TLS (Transport Layer Security) is the IETF standard version of SSL. TLS is very close to SSL.
- SSL provides security services for end-to-end applications.
- Most applications must be SSL enabled, i.e., SSL is **not** transparent.
- SSL is *algorithm independent*: for both public-key and symmetric-key operations, several algorithms are possible. Algorithms are negotiated on a per-session basis.

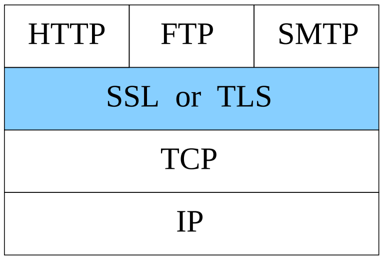


Figure 17.1: Location of SSL in the TCP/IP protocol stack.

- SSL consists of two main phases:

  Handshake Protocol : provides shared secret key using public-key techniques and mutual entity authentication.

  Record Protocol : provides confidentiality and message integrity for application data, using the shared secret established during the Handshake Protocol.

17.2 SSL Record Protocol

The SSL Record Protocol provides two main services:

1. *Confidentiality*: SSL payloads are encrypted with a symmetric cipher. The keys are for the symmetric cipher and they must be established during the preceding handshake protocol.

2. *Message Integrity*: the integrity of the message is provided through HMAC, a message authentication code.

17.2.1 Overview of the SSL Record Protocol

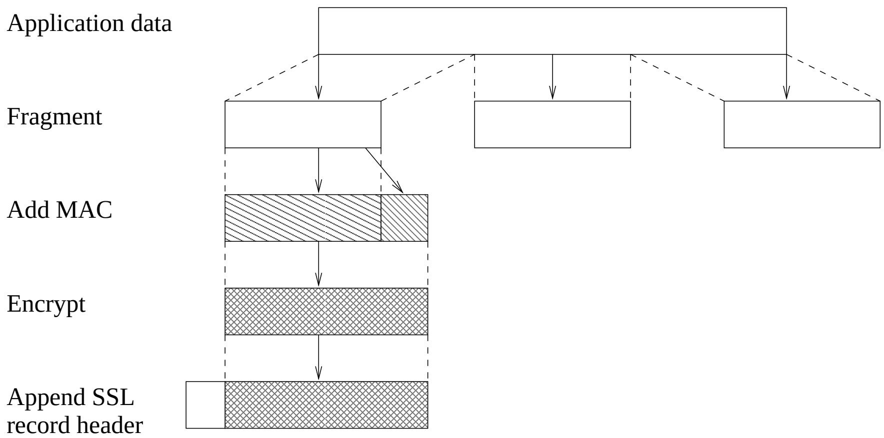


Figure 17.2: Simplified operations of the SSL Record Protocol

Description:

- **Fragmentation**: the message is devided into blocks of 2^{14} bytes.

- **MAC**: a derivative of the popular HMAC message authentication code. HMACs are based on hash functions.

  MAC = H(secret-key || pad2 ||

H(secret-key || pad1 || seq-num || fragment-length || fragment))

where:

H = hash algorithm; either MD5 or SHA-1.

secret-key = shared secret session key.

pad1 = the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1.

pad2 = the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1.

seq-num = the sequence number of the message.

fragment-length = length of the fragment (plaintext).

fragment = the plaintext block for which the MAC is computed.

- **Encrypt**: the following algorithms are allowed:

  1. Block ciphers:
     - IDEA (128-bit key)
     - RC-2 (40-bit key)
     - DES-40 (40-bit key)
     - DES (56-bit key)
     - 3DES (168-bit key)
     - Fortezza (80-bit key)
  2. Stream ciphers:
     - RC4-40 (40-bit key)
     - RC4-128 (128-bit key)

17.3 SSL Handshake Protocol

Remark: Most complex part of SSL, requires costly public-key operations

17.3.1 Core Cryptographic Components of SSL

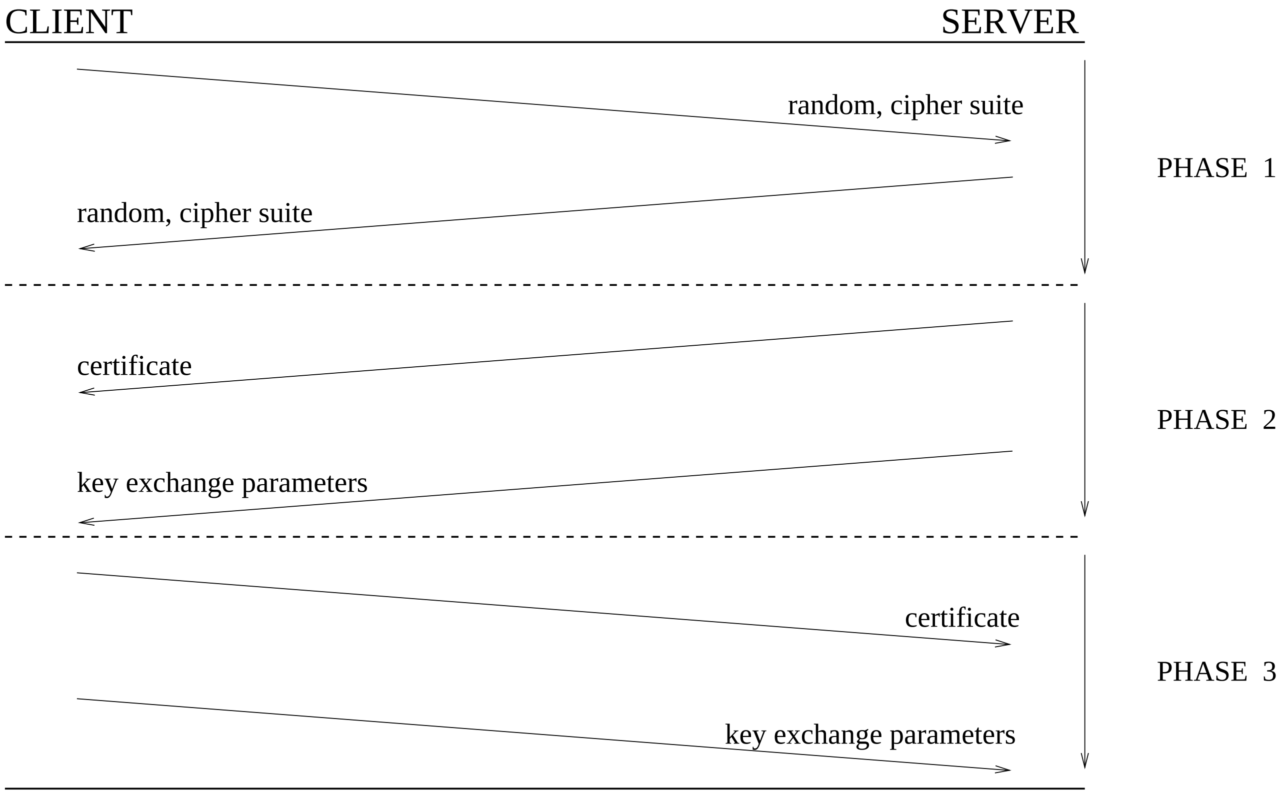


Figure 17.3: Simplified SSL Handshake Protocol

Explanation:

- **Phase 1**: establish security capabilities.

  random : 32-bit timestamp concatenated with 28-byte random value. Used as nonces and to prevent replay attacks during the key exchange.

  cipher suite : several fields, in particular:

1. Key exchange method.
   (a) RSA: the secret key is encrypted with the receiver's public RSA-key. Certificates are required.
   (b) Authenticated Diffie-Hellman: Diffie-Hellman with certificate.
   (c) Anonymous Diffie-Hellman: Diffie-Hellman without authentication.
   (d) Fortezza
2. Secret-key algorithm (see Section 17.2).
3. MAC algorithm (MD5 or SHA-1).

- **Phase 2**: server authentication and key exchange.

  Certificate : authenticated public key for any key exchange method except anonymous Diffie-Hellman.

  Key exchange parameters : signed public-key parameters, depending on the key exchange method.

- **Phase 3**: see Phase 2.

Chapter 18

Introduction to Identification Schemes

Examples for electronic identification situation:

1. Money withdrawal from ATM machine (PIN).
2. Credit card purchase over telephone (card number).
3. Remote computer login (user name and password).

Distinction between identification (or entity authentication) and message authentication:

- Identification schemes are performed online.
- Identification schemes do not require a meaningful message.

Basis for identification techniques:

$$\left.\begin{array}{l}\text{1. Something known (password, PIN)}\\ \text{2. Something possessed (chipcard)}\end{array}\right\}\text{cryptography based}$$

3. Something inherent to a human individual (fingerprint, retina pattern)

Overview:

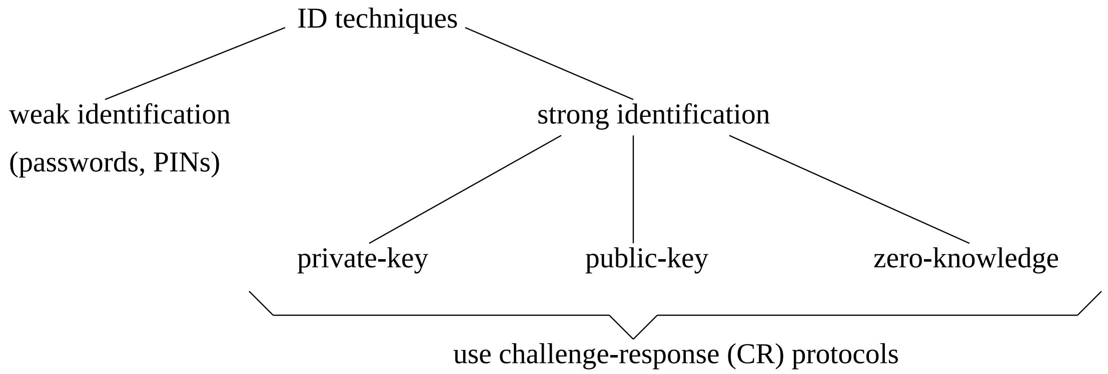


Figure 18.1: Identification Techniques

$\Rightarrow$ passwords and PINs are weak since they violate requirement 1 below.

Goals (informal definition):

1. Alice wants to prove her identity to Bob without revealing her identifying information to a listening Oscar. ("strong identification")

2. Also, Bob should not be able to impersonate Alice.

To achieve these goals, Alice has to perform a *proof of knowledge* which in general involves a *challenge-and-response* protocol.

18.1 Private-key Approach

Challenge-and-response (CR) protocol:

Assumption: Alice and Bob share a secret key k_{AB} and a keyed one-way function $f(x)$.

Alice		Bob
		1) generate challenge x
	$\xleftarrow{x}$	
2) $y = f_{k_{AB}}(x)$	$\xrightarrow{y}$	
		3) $y' = f_{k_{AB}}(x)$
		4) verification: $y \stackrel{?}{=} y'$

Example:

a) $f_k(x) = DES_k(x)$.

b) $f_k(x) = H(k||x)$.

c) $f_k(x) = x^k \bmod p$.

Remarks:

- CR protocols are standardized in ISO/IEC 9798.
- There are many variations to the above protocol, e.g., including time stamps or serial numbers in the response.
- Instead of block ciphers, public-key algorithms and keyed hash functions can be used.

Variant with time stamp (TS)

Alice		Bob
1) $y = e_{k_{AB}}(TS, ID(\text{Bob}))$	
	$\xrightarrow{y}$
		2) $(TS', ID'(\text{Bob}) = e^{-1}_{k_{AB}}(y)$
		$TS \overset{?}{\leq} \text{time} \overset{?}{\leq} TS + \epsilon$

Bibliography

[AM97] S.A. Vanstone A.J. Menezes, P.C. Oorschot. *Handbook of Applied Cryptography.* CRC Press, 1997.

[Big85] N.L. Biggs. *Discrete Mathematics.* Oxford University Press, New York, 1985.

[Bih97] E. Biham. A Fast New DES Implementation in Software. In *Fourth International Workshop on Fast Software Encryption* , volume LNCS 1267, pages 260–272, Berlin, Germany, 1997. Springer-Verlag.

[EYCP00] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. In *Third Advanced Encryption Standard (AES3) Conference*, pages 13–27, New York, USA, March 13–14, 2000. National Institute of Standards and Technology (NIST).

[Kob94] N. Koblitz. *A Course in Number Theory and Cryptography.* Springer-Verlag, New York, second edition, 1994.

[Men93] A.J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[OP00] Gerardo Orlando and Christof Paar. A High-Performance reconfigurable Elliptic Curve Processor for $GF(2^m)$. In Cetin K. Koc and Christof Paar, editors, *Cryp-*

tographic Hardware and Embedded Systems (CHES'2000), pages 41–56, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume.

[Sch93] B. Schneier. *Applied Cryptography.* Wiley & Sons, 1993.

[Sim92] G.J. Simmons. *Contemporary Cryptology.* IEEE Press, 1992.

[Sta95] W. Stallings. *Network and Internetwork Security.* Prentice Hall, 1995.

[Sta99] W. Stallings. *Cryptography and Network Security – Principles and Practice.* Prentice Hall, 2nd edition, 1999.

[Sti95] D.R. Stinson. *Cryptography, Theory and Practice.* CRC Press, 1995.

[WD76] M.E. Hellman W. Diffie. New directions in cryptography. In *IEEE Transactions on Information Theory*, volume IT-22, pages 644–654, 1976.

[WPR+99] D. Craig Wilcox, Lyndon G. Pierson, Perry J. Robertson, Edward L. Witzke, and Karl Gass. A DES ASIC Suitable for Network Encryption at 10 Gbps and Beyond. In Cetin K. Koc and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES'99)*, pages 37–48, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1717.

[WWGP00] T. Wollinger, M. Wang, J. Guajardo, and C. Paar. How Well Are High-End DSPs Suited for the AES Algorithms? AES Algorithms on the TMS320C6x DSP. In *Third Advanced Encryption Standard (AES3) Conference*, pages 94–105, New York, USA, March 13–14, 2000. National Institute of Standards and Technology (NIST).