

COMP547 Final Review Note

COMP547 Final Review Note

Ye Yuan — 260921269

December, 2022

COMP 547

Contents

1	Lecture1	4
2	Lecture2	4
3	Lecture3	4
4	Lecture4	10
5	Lecture5	14
6	Lecture6	22
7	Lecture7 and 8	28
8	Lecture9	34
9	Lecture10	43
10	Lecture11	47
11	Lecture12	52
12	Lecture13	57
13	Lecture14	71
14	Lecture15	82
15	Lecture16	88

16 Lecture17	94
17 Lecture18	101
18 Lecture19	105
19 Lecture20	106
20 Lecture21	111
21 Lecture22	118
22 Lecture23	123
23 Lecture24	130
24 Lecture25	138
25 Lecture26	138

1 Lecture1

Introduction

2 Lecture2

Introduction

3 Lecture3

Long Addition: $O(|a| + |b|)$

Long Subtraction: $O(|a| + |b|)$

Long Multiplication: $O(|a| \cdot |b|)$

Long Division: $O(|a| \cdot |b|)$

Divisibility: $a|b \iff \exists k \in \mathbb{Z} \text{ s.t. } b = ak$

Congruence: $a \equiv b \pmod{n} \iff n|(a - b)$

Modulo Operator: $b \bmod n = \min\{a \geq 0 : a \equiv b \pmod{n}\}$

Division Operator: $b \operatorname{div} n = \operatorname{floor}\left(\frac{b}{n}\right) = \frac{b - (b \bmod n)}{n}$

$a \equiv b \pmod{n} \iff a \bmod n = b \bmod n$

THEOREM B.1 (Integer operations) Given integers a and b , it is possible to perform the following operations in time polynomial in $\|a\|$ and $\|b\|$:

1. Computing the sum $a + b$ and the difference $a - b$;
2. Computing the product ab ;
3. Computing positive integers q and $r < b$ such that $a = qb + r$ (i.e., computing division with remainder);
4. Computing the greatest common divisor of a and b , $\gcd(a, b)$;
5. Computing integers X, Y with $Xa + Yb = \gcd(a, b)$.

Figure 1

Greatest Common Divisor: $g = \gcd(a, b) \iff g|a, g|b$ and $[g'|a, g'|b \implies g'|g]$

Euler's Phi Function: $\Phi(N) = \#\{a : 0 < a < N \text{ and } \gcd(a, N) = 1\}$

- when p is prime, $\Phi(p) = p - 1$
- when p and q are primes, $\Phi(pq) = (p - 1)(q - 1)$
- when $N = p_1^{e_1} \dots p_k^{e_k}$ is a product of distinct prime powers, $\Phi(N) = (p_1 - 1)p_1^{e_1 - 1} \dots (p_k - 1)p_k^{e_k - 1}$

ALGORITHM B.7

The Euclidean algorithm GCD

Input: Integers a, b with $a \geq b > 0$

Output: The greatest common divisor of a and b

if b divides a

return b

else return $\text{GCD}(b, [a \bmod b])$

Figure 2: Running Time: $O(|a| \cdot |b|)$

ALGORITHM B.10**The extended Euclidean algorithm eGCD****Input:** Integers a, b with $a \geq b > 0$ **Output:** (d, X, Y) with $d = \gcd(a, b)$ and $Xa + Yb = d$ **if** b divides a **return** $(b, 0, 1)$ **else**Compute integers q, r with $a = qb + r$ and $0 < r < b$ $(d, X, Y) := \text{eGCD}(b, r)$ // note that $Xb + Yr = d$ **return** $(d, Y, X - Yq)$

$r := a \bmod b$ $q := a \div b$

Figure 3: Running Time: $O(|a| \cdot |b|)$

In Figure. 3, we have $d = \gcd(b, r) = Xb + Yr$, with $r = a - qb$. As well as $d = \gcd(a, b) = Xb + aY - qbY = Ya + (X - Yq)b$

Example

$\gcd(19, 7)$	$d=1, q=2, X=3, Y=-8$
$= \gcd(7, 19 \bmod 7)$	
$= \gcd(7, 5)$	$d=1, q=1, X=-2, Y=3$
$= \gcd(5, 7 \bmod 5)$	
$= \gcd(5, 2)$	$d=1, q=2, X=1, Y=-2$
$= \gcd(2, 5 \bmod 2)$	
$= \gcd(2, 1)$	$d=1, X=0, Y=1$
$= 1$	

THEOREM B.2 (Modular operations) *Given integers $N > 1$, a , and b , it is possible to perform the following operations in time polynomial in $\|a\|$, $\|b\|$, and $\|N\|$:*

1. *Computing the modular reduction $[a \bmod N]$;*
2. *Computing the sum $[(a+b) \bmod N]$, the difference $[(a-b) \bmod N]$, and the product $[ab \bmod N]$;*
3. *Determining whether a is invertible modulo N ;*
4. *Computing the multiplicative inverse $[a^{-1} \bmod N]$, assuming a is invertible modulo N ;*
5. *Computing the exponentiation $[a^b \bmod N]$.*

Figure 4

The following rule is used to reduce the workload:

$$a \ + \ - \ \times \ b \bmod N = ((a \bmod N) \ + \ - \ \times \ (b \bmod N)) \bmod N$$

ALGORITHM B.11

Computing modular inverses

Input: Modulus N ; element a

Output: $[a^{-1} \bmod N]$ (if it exists)

$(d, X, Y) := \text{eGCD}(a, N)$ // note that $Xa + YN = \text{gcd}(a, N)$

if $d \neq 1$ **return** “ a is not invertible modulo N ”

else return $[X \bmod N]$

Figure 5: Running Time: $O(|N|^2)$

Example

$$\text{gcd}(19, 7)$$

$$= \text{gcd}(2, 1) = 1 \qquad x=-8, y=3$$

thus,

$$1 = -8 \times 7 + 3 \times 19$$

and,

$$7^{-1} \bmod 19 = 11 = -8 \bmod 19$$

$$19^{-1} \bmod 7 = 3 = 3 \bmod 7$$

ALGORITHM B.13**Algorithm ModExp for efficient modular exponentiation****Input:** Modulus N ; base $a \in \mathbb{Z}_N$; integer exponent $b > 0$ **Output:** $[a^b \bmod N]$ $x := a$ $t := 1$ // maintain the invariant that the answer is $[t \cdot x^b \bmod N]$ **while** $b > 0$ **do:** **if** b is odd $t := [t \cdot x \bmod N]$, $b := b - 1$ $x := [x^2 \bmod N]$, $b := b/2$ **return** t *Figure 6: Running Time: $O(|a| \cdot |b| \cdot |N|)$* **Example**

$$5^{13} \bmod 7$$

$$= 5^{8+4+1} \bmod 7$$

$$= 5^8 \times 5^4 \times 5^1 \bmod 7$$

$$5^0, 5^1, 5^2, 5^4, 5^8 \equiv 1, 5, 4, 2, 4 \pmod{7}$$

$$= 4 \times 2 \times 5 \bmod 7$$

$$= 1 \times 5 \bmod 7$$

$$= 5$$

Solving linear congruentials: A linear congruential is an expression of the form $ax \equiv b \pmod{N}$ for known a, b, N and unknown x . Clearly, we can solve for x whenever $\gcd(a, N) = 1$ since in that case $a^{-1} \pmod{N}$ exists and $x \equiv b \cdot a^{-1} \pmod{N}$

Example

$$7x \equiv 5 \pmod{9}$$

$$7^{-1} \cdot 7x \equiv 7^{-1} \cdot 5 \pmod{9}$$

$$4 \cdot 7x \equiv 4 \cdot 5 \pmod{9}$$

$$x \equiv 2 \pmod{9}$$

To solve $b \equiv ax \pmod{N}$

- Compute $g = \gcd(a, N)$
- If $g \nmid b$ then there are no solutions
- Otherwise, there are g distinct solutions for $0 \leq k < g$, given by $x_0 = \hat{x}, \dots, x_k = \hat{x} + k\hat{N}, \dots, x_{g-1} = \hat{x} + (g-1)\hat{N}$, where $\hat{N} = \frac{N}{g}, \hat{a} = \frac{a}{g}, \hat{b} = \frac{b}{g}, \hat{x} \equiv \hat{b} \cdot \hat{a}^{-1} \pmod{\hat{N}}$

Example

$$6x \equiv 5 \pmod{9}$$

has no solution because $\gcd(6, 9) = 3$ and $3 \nmid 5$

$$6x \equiv 3 \pmod{9}$$

$$2x \equiv 1 \pmod{3} \quad \text{since } \gcd(6, 9) | 3$$

$$2^{-1} \cdot 2x \equiv 2^{-1} \cdot 1 \pmod{3}$$

$$x \equiv 2 \pmod{3}$$

$$x \equiv 2, 5, 8 \pmod{9}$$

4 Lecture4

THEOREM 8.24 (Chinese remainder theorem) *Let $N = pq$ where $p, q > 1$ are relatively prime. Then*

$$\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q \quad \text{and} \quad \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*.$$

Moreover, let f be the function mapping elements $x \in \{0, \dots, N-1\}$ to pairs (x_p, x_q) with $x_p \in \{0, \dots, p-1\}$ and $x_q \in \{0, \dots, q-1\}$ defined by

$$f(x) \stackrel{\text{def}}{=} ([x \bmod p], [x \bmod q]).$$

Then f is an isomorphism from \mathbb{Z}_N to $\mathbb{Z}_p \times \mathbb{Z}_q$, and the restriction of f to \mathbb{Z}_N^ is an isomorphism from \mathbb{Z}_N^* to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.*

Figure 7

Chinese Remainder Theorem:

Let m_1, m_2, \dots, m_r be r positive integers such that $\gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq r$ and let a_1, a_2, \dots, a_r be arbitrary integers.

The system of r congruences,

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ \dots \\ x \equiv a_r \pmod{m_r} \end{cases}$$

has a unique solution modulo $M = m_1 \dots m_r$ given by

$$x = \sum_{i=1}^r a_i M_i y_i \pmod{M}$$

where $M_i = \frac{M}{m_i}$, and $y_i = M_i^{-1} \pmod{m_i}$

Example

$$\begin{cases} x \equiv 5 \pmod{9} \\ x \equiv 3 \pmod{4} \\ x \equiv 7 \pmod{13} \end{cases}$$

has a unique solution mod $468 = 4 \times 9 \times 13$

$$x = 5 \cdot 52 \cdot (52^{-1} \bmod 9) + 3 \cdot 117 \cdot (117^{-1} \bmod 4) + 7 \cdot 36 \cdot (36^{-1} \bmod 13)$$

$$x = 260 \cdot 4 + 351 \cdot 1 + 252 \cdot 4$$

$$x = 1040 + 351 + 1008$$

$$x = 2399$$

$$x \equiv 59 \pmod{468}$$

Quadratic Residues Quadratic residues modulo N are the integers with an integer square root modulo N :

- $QR_N = \{a : \gcd(a, N) = 1, \exists r[a \equiv r^2 \pmod{N}]\}$
- $QNR_N = \{a : \gcd(a, N) = 1, \forall r[a \not\equiv r^2 \pmod{N}]\}$

Example:

$$QR_{17} = \{1, 2, 4, 8, 9, 13, 15, 16\}$$

$$QNR_{17} = \{3, 5, 6, 7, 10, 11, 12, 14\}$$

$$\text{since } \{1^2, 2^2, 3^2, 4^2, 5^2, 6^2, 7^2, 8^2, 9^2, 10^2, 11^2, 12^2, 13^2, 14^2, 15^2, 16^2\}$$

$$\equiv \{1, 2, 4, 8, 9, 13, 15, 16\} \pmod{17}$$

Theorem

Let p be an odd prime number

$$\#QR_p = \#QNR_p = \frac{p-1}{2}$$

Legendre Symbols

For an odd prime number p , we define the Legendre symbol as:

$$\left(\frac{a}{p}\right) = \begin{cases} +1 & \text{if } a \in QR_p \\ -1 & \text{if } a \in QNR_p \\ 0 & \text{if } p|a \end{cases}$$

Jacobi Symbols

For any integer $N = p_1 p_2 \dots p_k$, we define the Jacobi symbol (a generalization of the Legendre symbol) as

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\dots\left(\frac{a}{p_k}\right)$$

Properties of Legendre and Jacobi Symbols

- $\left(\frac{1}{N}\right) = +1$
- $\left(\frac{ab}{N}\right) = \left(\frac{a}{N}\right)\left(\frac{b}{N}\right)$
- $\left(\frac{a}{N}\right) = \left(\frac{a \bmod N}{N}\right)$
- $\left(\frac{-1}{N}\right) = (-1)^{(N-1)/2}$ for N odd
- $\left(\frac{2}{N}\right) = (-1)^{(N^2-1)/8}$ for N odd
- $\left(\frac{a}{N}\right)\left(\frac{N}{a}\right) = (-1)^{(N-1)(a-1)/4}$ for a, N odd and such that $\gcd(a, N) = 1$

ALGORITHM B.ω

Jacobi Symbols Algorithm Jaco

Input: non-negative Integers a, b

Output: $\left(\frac{a}{b}\right)$ the Jacobi Symbol of a w.r.t. b

if $a \leq 1$ **return** a

if a is odd **if** $a \equiv b \equiv 3 \pmod{4}$ **return**

$-\text{Jaco}(b \bmod a, a)$

else return $+\text{Jaco}(b \bmod a, a)$

else if $b \equiv \pm 1 \pmod{8}$ **return** $+\text{Jaco}(a/2, b)$

else return $-\text{Jaco}(a/2, b)$

Figure 8: Running Time: $O(|a| \cdot |b|)$

Example

Jacobi(527, 723)

= -Jacobi(723 mod 527, 527)

$$= -\text{Jacobi}(196, 527)$$

$$= -\text{Jacobi}(98, 527)$$

$$= -\text{Jacobi}(49, 527)$$

$$= -\text{Jacobi}(527 \bmod 49, 49)$$

$$= -\text{Jacobi}(37, 49)$$

$$= -\text{Jacobi}(49 \bmod 37, 37)$$

$$= -\text{Jacobi}(12, 37)$$

$$= \text{Jacobi}(6, 37)$$

$$= -\text{Jacobi}(3, 37)$$

$$= -\text{Jacobi}(37 \bmod 3, 3)$$

$$= -\text{Jacobi}(1, 3)$$

$$= -1$$

Theorem [Fermat]

Let p be a prime and a be an integer not a multiple of p , then

$$a^{p-1} \equiv 1 \pmod{p}$$

Theorem [Euler]

Let p be a prime number and a be an integer, then

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod{p}$$

Theorem [Euler]

Let N be an integer and a another integer such that $\gcd(a, N) = 1$, then

$$a^{\Phi(N)} \equiv 1 \pmod{N}$$

5 Lecture5

ALGORITHM 15.31

Computing square roots modulo a prime

Input: Prime p ; quadratic residue $a \in \mathbb{Z}_p^*$

Output: A square root of a

```

let  $b$  be a quadratic non-residue modulo  $p$ 
compute  $\ell \geq 1$  and odd  $m$  with  $2^\ell \cdot m = \frac{p-1}{2}$ 
 $r := 2^\ell \cdot m, r' := 0$ 
for  $i = \ell$  to 1 {
  // maintain the invariant  $a^r \cdot b^{r'} = 1 \pmod p$ 
   $r := r/2, r' := r'/2$ 
  if  $a^r \cdot b^{r'} = -1 \pmod p$ 
     $r' := r' + 2^\ell \cdot m$ 
}
// now  $r = m$ ,  $r'$  is even, and  $a^r \cdot b^{r'} = 1 \pmod p$ 
return  $\left[ a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \pmod p \right]$ 

```

Figure 9

ALGORITHM 15.31

Computing square roots modulo a prime

Input: Prime p ; quadratic residue $a \in \mathbb{Z}_p^*$

Output: A square root of a

```

let  $b$  be a quadratic non-residue modulo  $p$ 
compute  $\ell \geq 1$  and odd  $m$  with  $2^\ell \cdot m = \frac{p-1}{2}$ 
 $r := \frac{p-1}{2}, r' := 0$ 
while  $r$  is even do{
  // maintain the invariant  $1 = a^r \cdot b^{r'} \pmod p$ 
   $r := r/2, r' := r'/2$ 
  if  $p-1 = a^r \cdot b^{r'} \pmod p$ 
     $r' := r' + \frac{p-1}{2}$ 
}
// now  $r = m$ ,  $r'$  is even, and  $1 = a^r \cdot b^{r'} \pmod p$ 
return  $\left[ a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \pmod p \right]$ 

```

Figure 10

Example

$p = 43, a = 16$

$$b = -1$$

$$\text{return } \pm 16^{11} \bmod 43 = (\pm 4)$$

$$p = 37, a = 16$$

$$b = 2$$

$$r := (p-1)/2 = 18, r' := 0$$

$$r := r/2 = 9, r' := 0$$

$$\text{return } \pm 16^5 \bmod 37 (= \pm 4)$$

$$p = 41, a = 16$$

$$b = 3$$

$$r := (p-1)/2 = 20, r' := 0$$

$$r := r/2 = 10, r' := 0$$

$$r := r/2 = 5, r' := 0$$

$$\text{return } \pm 16^3 \bmod 41 (= \pm 4)$$

$$p = 41, a = 8$$

$$b = 3$$

$$r := (p-1)/2 = 20, r' := 0$$

$$r := r/2 = 10, r' := r'/2 + 20 = 20$$

$$r := r/2 = 5, r' := r'/2 + 20 = 30$$

$$\text{return } \pm 8^3 \times 3^{15} \bmod 41 (= \pm 7)$$

Extracting Square Roots modulo N

We want to solve $r^2 \equiv a \pmod{N}$ for r knowing p, q such that $N = pq$. We first solve modulo p and q and find solutions to

$$\begin{cases} r_p^2 \equiv a \pmod{p} \\ r_q^2 \equiv a \pmod{q} \end{cases}$$

We then consider the simultaneous congruences

$$r \equiv r_p \pmod{p} \iff p \mid r^2 - a$$

$$r \equiv r_q \pmod{q} \iff q \mid r^2 - a$$

$$\implies p \cdot q = N \mid r^2 - a$$

$$\implies r^2 \equiv a \pmod{N}$$

We can now solve r using the Chinese Remainder Theorem.

Example

$$N = 37 \times 43 = 1591, a = 16$$

$$\sqrt{a} \equiv \pm 33 \pmod{37}$$

$$\sqrt{a} \equiv \pm a^{(p+1)/4} \equiv \pm a^{11} \equiv \pm 4 \pmod{43}$$

Use the Chinese Remainder Theorem on the following 4 systems:

$$\begin{cases} \sqrt{a} \equiv 4 \pmod{37}, \sqrt{a} \equiv 4 \pmod{43} \\ \sqrt{a} \equiv 4 \pmod{37}, \sqrt{a} \equiv 39 \pmod{43} \\ \sqrt{a} \equiv 33 \pmod{37}, \sqrt{a} \equiv 4 \pmod{43} \\ \sqrt{a} \equiv 33 \pmod{37}, \sqrt{a} \equiv 39 \pmod{43} \end{cases}$$

$$1. \sqrt{a} \equiv 4 \pmod{37}, \sqrt{a} \equiv 4 \pmod{43}$$

$$\sqrt{a} \equiv 4 \pmod{1591}$$

$$2. \sqrt{a} \equiv 4 \pmod{37}, \sqrt{a} \equiv 39 \pmod{43}$$

$$\sqrt{a} \equiv 1114 \pmod{1591}$$

$$3. \sqrt{a} \equiv 33 \pmod{37}, \sqrt{a} \equiv 4 \pmod{43}$$

$$\sqrt{a} \equiv 477 \pmod{1591}$$

$$4. \sqrt{a} \equiv 33 \pmod{37}, \sqrt{a} \equiv 39 \pmod{43}$$

$$\sqrt{a} \equiv 1587 \pmod{1591}$$

Definition [SQROOT]

The square root modulo N problem is: given a composite integer N and $a \in \mathbb{Q}R_N$, find an integer r , $0 \leq r < N$, such that $a \equiv r^2 \pmod{N}$.

Theorem:

SQROOT is polynomially equivalent to FACTORING.

Proof idea:

The previous construction shows that if we know the factorization of N , we can extract square roots modulo each prime factor and then recombine using the Chinese Remainder Theorem. If we can extract square roots modulo N , we can split N in two factors $N = uv$ using the following algorithm, and by repetition fully factor it: The probability of the else case is at least $1/2$.

ALGORITHM B.ω+1

Integer Splitting

Input: non-negative Integer N

Output: Integers $u, v > 1$ s.t. $N = uv$

$r \leftarrow \{1 \dots N-1\}$, $r' \leftarrow \mathbf{SQROOT}(r^2, N)$

if $r' \equiv \pm r \pmod{N}$ **return** **Splitting**(N)

else $u := \gcd(r + r', N)$, $v := \gcd(r - r', N)$

return (u, v)

Figure 11

```

ALGORITHM B.ω+2
Integer Factoring

Input: non-negative Integer  $N$ 
Output: List of Prime Integers  $p_1, \dots, p_k$  s.t.  $N = p_1 \cdots p_k$ 

if  $N$  is prime return  $[N]$ 
else  $(u, v) := \text{Splitting}(N)$ ; return Factoring( $u$ ) || Factoring( $v$ )

```

Figure 12

Example

$$N = 37 \times 43 = 1591$$

$$a = 477$$

$$a^2 \bmod 1591 = 16$$

$$\sqrt{(a^2)} \bmod 1591 = 4$$

$$477 \not\equiv \pm 4 \pmod{1591}$$

$$u = \gcd(477+4, 1591) = 37$$

$$v = \gcd(477-4, 1591) = 43$$

Prime Numbers

If we want a random prime of a given size, we use the following theorem to estimate the number of integers we must try before finding a prime. Let $\pi(N) = \#\{a : 1 < a \leq N \text{ and } a \text{ is prime}\}$

Theorem

$$\lim_{N \rightarrow \infty} \frac{\pi(N) \log N}{N} = 1$$

To decide whether a number N is prime or not we rely on Miller-Rabin's probabilistic algorithm. This algorithm introduces the notion of "pseudo primality" base a . Miller defined this test as an extension of Fermat's test.

If the Extended Riemann Hypothesis is true then it is sufficient to use the test with small values of a to decide whether a number N is prime or composite. However the ERH is not proven and we use the test in a probabilistic fashion as suggested by Rabin.

ALGORITHM 8.31

Generating a random prime – high-level outline

Input: Length n ; parameter t

Output: A uniform n -bit prime

```

for  $i = 1$  to  $t$ :
     $p' \leftarrow \{0,1\}^{n-2}$ 
     $p := 1 \parallel p' \parallel 1$ 
    if  $p$  is prime return  $p$ 
return fail
  
```

Figure 13

ALGORITHM 8.34

Generating a random prime

Input: Length n

Output: A uniform n -bit prime

```

for  $i = 1$  to  $3n^2$ :
     $p' \leftarrow \{0,1\}^{n-2}$ 
     $p := 1 \parallel p' \parallel 1$ 
    run the Miller–Rabin test on input  $p$  and parameter  $1^n$ 
    if the output is “prime,” return  $p$ 
return fail
  
```

Figure 14

ALGORITHM 8.35

Primality testing – first attempt

Input: Integer N and parameter 1^t

Output: A decision as to whether N is prime or composite

```

for  $i = 1$  to  $t$ :
     $a \leftarrow \{1, \dots, N-1\}$ 
    if  $a^{N-1} \neq 1 \bmod N$  return “composite”
return “prime”
  
```

Figure 15

ALGORITHM 8.44**The Miller–Rabin primality test****Input:** Integer $N > 2$ and parameter 1^t **Output:** A decision as to whether N is prime or compositeif N is even, **return** “composite”if N is a perfect power, **return** “composite”**compute** $r \geq 1$ and u odd such that $N - 1 = 2^r u$ **for** $j = 1$ to t : $a \leftarrow \{1, \dots, N - 1\}$ **if** $a^u \not\equiv \pm 1 \pmod{N}$ and $a^{2^i u} \not\equiv -1 \pmod{N}$ for $i \in \{1, \dots, r - 1\}$ **return** “composite”**return** “prime”*Figure 16: Miller-Rabin Test*

As shown in Figure. 16, it is easy to show that if N is prime, then Miller-Rabin(N , t) returns “prime” with probability 1. Rabin showed that if N is composite, then Miller-Rabin(N , t) returns “prime” with probability at most 4^{-t} .

Example

$$N = 37 \times 43 = 1591$$

$$N-1 = 1590 = 795 \times 2$$

$$u = 795, r = 1$$

$$a = 16$$

$$a^u = 16^{795} \pmod{1591}$$

$$i = 0, a^u = 692 \neq 1$$

$$i = 1, a^{2u} = 1564 \neq 1 \quad \text{fail to satisfy the Fermat Theorem}$$

1591 is definitely composite

$$N = 1597$$

$$N - 1 = 1596 = 399 \times 4$$

$$u = 399, r = 2$$

$$a = 16$$

$$a^u = 16^{399} \bmod 1597$$

$$i = 0, a^u = 1$$

1597 is potentially prime

$$N = 1597$$

$$N - 1 = 1596 = 399 \times 4$$

$$u = 399, r = 2$$

$$a = 17$$

$$a^u = 17^{399} \bmod 1597$$

$$i = 0, a^u = 1$$

1597 is potentially prime

$$N = 1597$$

$$N - 1 = 1596 = 399 \times 4$$

$$u = 399, r = 2$$

$$a = 18$$

$$a^u = 18^{399} \bmod 1597$$

$$i = 0, a^u = 610$$

$$i = 1, a^{2u} = 1596$$

$$i = 2, a^{4u} = 1$$

1597 is potentially prime

$N = 1597$ is either prime or we just experienced an event (3 random occurrences of "prime") that happens with probability at most $1/64$. We are almost certain that 1597 is prime...

Deterministic Primality Testing

In August of 2002, Agrawal, Kayal, and Saxena, announced the discovery of a deterministic primality test running in polynomial time. Unfortunately this test is too slow in practice... its running time being $O(|N|^{12})$. In 2005, Pomerance and Lenstra demonstrated a faster variant of AKS that runs in $\tilde{O}(|N|^6)$ operations, but is it still rather slow in practice. To prove the primality of an integer N , N is prime if and only if for all a such that $\gcd(a, N) = 1$

$$(x + a)^N \equiv x^N + a \pmod{N}$$

The idea is to validate this (exponentially long) congruence in poly-time.

6 Lecture6

Prime Fields

Let p be a prime number. The integers $0, 1, 2, \dots, p-1$ with operations $+$ mod p and \times mod p constitute a field \mathcal{F}_p of p elements

- contains an additive neutral element (0)
- each element has an additive inverse $-e$
- contains an multiplicative neutral element (1)
- each non-zero element e has a multiplicative inverse e^{-1}
- associativity
- commutativity
- distributivity

Example

$$\mathcal{F}_2 = (\{0, 1\}, \oplus, \wedge)$$

$$\mathcal{F}_5 = (\{0, 1, 2, 3, 4\}, + \pmod{5}, \times \pmod{5}) \text{ defined by}$$

$+$	0	1	2	3	4	\times	0	1	2	3	4
0	0	1	2	3	4	0	0	0	0	0	0
1	1	2	3	4	0	1	0	1	2	3	4
2	2	3	4	0	1	2	0	2	4	1	3
3	3	4	0	1	2	3	0	3	1	4	2
4	4	0	1	2	3	4	0	4	3	2	1

Other kind of finite fields for numbers $q = p^e, e > 1$. We will not study them this term. In general we refer to \mathcal{F}_q for a finite field, but you may think of the special case \mathcal{F}_p if you do not wish to find out about the general field construction. Galois proved that if N is not a prime power, no field of N elements can be constructed.

$$\mathcal{F}_4 = (\{0, 1, 2, 3\}, +, \times) \text{ defined by}$$

$+$	0	1	2	3	\times	0	1	2	3
0	0	1	2	3	0	0	0	0	0
1	1	0	3	2	1	0	1	2	3
2	2	3	0	1	2	0	2	3	1
3	3	2	1	0	3	0	3	1	2

Note: $+$ is not $+$ (mod 4) and \times is not \times (mod 4)

Primitive Elements

In all finite fields \mathcal{F}_q (and some groups in general) there exists a primitive element, that is an element g of the field such that

$$g^1, g^2, \dots, g^{q-1}$$

enumerate all of the $q-1$ non-zero elements of the field. We use the following theorem to efficiently find a primitive element over \mathcal{F}_q .

Theorem

Let l_1, l_2, \dots, l_k be the prime factors of $q-1 = l_1 \cdot l_2 \dots l_k$ and

$$m_1 = \frac{q-1}{l_1}, m_2 = \frac{q-1}{l_2}, \dots, m_k = \frac{q-1}{l_k}$$

.

An element g is primitive over \mathcal{F}_q if and only if

- $g^{q-1} = 1$
- $g^{m_i} \neq 1$ for $1 \leq i \leq k$

ALGORITHM B.9.3 (related to Section 9.3)

Random Primitive Element of \mathbb{F}_q

Input: positive Integers q, l_1, \dots, l_k (prime factors of $q-1$)

Output: Primitive Element $g \in \mathbb{F}_q^*$

Let $m_1 = \frac{q-1}{l_1}, \dots, m_k = \frac{q-1}{l_k}$.

repeat

$g \leftarrow \mathbb{F}_q^*$

until $\forall i, 1 \leq i \leq k, g^{m_i} \neq 1$

return $g \in \mathbb{F}_q^*$

Figure 17: \mathcal{F}_q^* is not including 0

We use the following theorems to estimate the number of field elements. We must try in order to find a random primitive element.

Theorem

$$\# \{g: g \text{ is a primitive element of } \mathcal{F}_q\} = \Phi(q-1)$$

Theorem

$$\lim_{N \rightarrow \infty} \inf \frac{\Phi(N) \log \log N}{N} = e^{-\gamma} \approx 0.5614594836$$

Example

2 is a primitive element of \mathcal{F}_5 since $\{2, 2^2, 2^3, 2^4\} = \{2, 4, 3, 1\}$.

Relation to Quadratic Residues

As an interesting note, if g is a primitive element of the prime field \mathcal{F}_p then we have:

- $QR_p = \{g^{2i} \bmod p : 0 \leq i < (p-1)/2\}$
- $QNR_p = \{g^{2i+1} \bmod p : 0 \leq i < (p-1)/2\}$

In other words, the quadratic residues are the even powers of g , while the quadratic non-residues are the odd powers of g .

Polynomials over a field

A polynomial over \mathcal{F}_p is specified by a finite sequence $(a_n, a_{n-1}, \dots, a_1, a_0)$ of elements from \mathcal{F}_p , with $a_n \neq 0$. The number n is the degree of the polynomial. We have operations $+$, $-$, \times on polynomials analogous to the similar integer operations. Addition and subtraction are performed component-wise using the addition $+$ and subtraction $-$ of the field \mathcal{F}_p . Products are computed by adding all the products of coefficients associated to pairs of exponents adding to a specific exponent.

Example over \mathcal{F}_2

$$\begin{aligned}
 & (x^4 + x + 1) \times (x^3 + x^2 + x) \\
 &= x^4 \times (x^3 + x^2 + x) + x \times (x^3 + x^2 + x) + 1 \times (x^3 + x^2 + x) \\
 &= (x^7 + x^6 + x^5) + (x^4 + x^3 + x^2) + (x^3 + x^2 + x) \\
 &= x^7 + x^6 + x^5 + x^4 + (1+1)x^3 + (1+1)x^2 + x \\
 &= x^7 + x^6 + x^5 + x^4 + x
 \end{aligned}$$

We also have operations $g(x) \bmod H(x)$ and $g(x) \operatorname{div} H(x)$ defined as the unique polynomials $r(x)$ and $q(x)$ such that $g(x) = q(x)H(x) + r(x)$ with $\deg(r) < \deg(H)$. They are obtained by formal division of $g(x)$ by $H(x)$ similar to what we do with integers.

Example over \mathcal{F}_2

$$x^7 + x^6 + x^5 + x^4 + x$$

$$= (x^2) \times (x^5 + x^2 + 1) + (x^6 + x^5 + x^2 + x)$$

$$= (x^2 + x) \times (x^5 + x^2 + 1) + (x^5 + x^3 + x^2)$$

$$= (x^2 + x + 1) \times (x^5 + x^2 + 1) + (x^3 + 1)$$

thus

$$(x^7 + x^6 + x^5 + x^4 + x) \bmod (x^5 + x^2 + 1) = x^3 + 1$$

$$(x^7 + x^6 + x^5 + x^4 + x) \operatorname{div} (x^5 + x^2 + 1) = x^2 + x + 1$$

Exponentiations for integer powers modulo a polynomial are computed using an analogue of algorithm B.13 and gcd of polynomials or multiplicative inverses are computed using an analogue of algorithm B.10.

Irreducible Polynomials over a field

A polynomial $g(x)$ is irreducible if it is not the product of two polynomials $H(x)$, $k(x)$ of lower degrees. We use the following theorem to find irreducible polynomials.

Theorem

Let l_1, l_2, \dots, l_k be the prime factors of n and $m_i = n/l_i$ for $1 \leq i \leq k$. A polynomial $g(x)$ of degree n is irreducible over \mathcal{F}_p if and only if

- $g(x) \mid x^{p^n} - x$
- $\gcd(g(x), x^{p^{m_i}} - x) = 1$ for $1 \leq i \leq k$

Examples

1. \mathcal{F}_2 $x + 1, x^2 + x + 1, x^3 + x + 1, x^4 + x + 1, x^5 + x^2 + 1, x^6 + x + 1, x^7 + x^3 + 1, x^8 + x^4 + x^3 + x^2 + 1, x^9 + x^4 + 1, x^{10} + x^3 + 1, x^{11} + x^2 + 1, x^{12} + x^6 + x^4 + x + 1, x^{13} + x^4 + x^3 + x + 1, x^{14} + x^{10} + x^6 + x + 1, x^{15} + x + 1, x^{16} + x^{12} + x^3 + x + 1$
2. \mathcal{F}_3 $x + 1, x^2 + x + 2, x^3 + 2x + 1, x^4 + x + 2, x^5 + 2x + 1, x^6 + x + 2$
3. \mathcal{F}_5 $x + 1, x^2 + x + 2, x^3 + 3x + 2, x^4 + x^2 + x + 2$
4. \mathcal{F}_7 $x + 1, x^2 + x + 3, x^3 + 3x + 2$

ALGORITHM A.ω+5 (related to Section A.5)

Random Irreducible Polynomial over \mathbb{F}_q

Input: positive Integers n, p (a prime)

Output: Irreducible Polynomial g over \mathbb{F}_q

Let l_1, \dots, l_k be the prime factors of $n - 1$
 and $m_1 = \frac{n-1}{l_1}, \dots, m_k = \frac{n-1}{l_k}$.

repeat
 pick a random polynomial $h(x)$ of degree $n - 1$ over \mathbb{F}_p ,
Let $g(x) = x^n + h(x)$,
until $x^{p^n} \bmod g(x) = x$ **and**
 $\forall i, 1 \leq i \leq k, \gcd(g(x), x^{p^{m_i}} \bmod g(x) - x) = 1$
return g




Figure 18

We use the following theorem to estimate the number of polynomials we have to try on average before finding one that is irreducible.

Theorem

Let $m(n)$ be the number of irreducible polynomials $g(x)$ of degree n of the form $g(x) = x^n + H(x)$ where $H(x)$ is of degree $n-1$. We have

$$\frac{p^n}{2n} \leq m(n) \leq \frac{p^n}{n}$$

General Fields

Let p be a prime number and n a positive integer. We construct a field with p^n elements from the basis field \mathcal{F}_p with p elements.

- The elements of \mathcal{F}_{p^n} are of the form $a_1 a_2 \dots a_n$ where $a_i \in \mathcal{F}_p$
- The sum of two elements of \mathcal{F}_{p^n} is defined by $a_1 a_2 \dots a_n + b_1 b_2 \dots b_n = c_1 c_2 \dots c_n$ such that $c_i = a_i + b_i \bmod p$ for $1 \leq i \leq n$
- The product of two elements of \mathcal{F}_{p^n} is defined by $a_1 a_2 \dots a_n \times b_1 b_2 \dots b_n = c_1 c_2 \dots c_n$ such that $(c_1 x^{n-1} + \dots + c_n) = (a_1 x^{n-1} + \dots + a_n) \times (b_1 x^{n-1} + \dots + b_n) \bmod r(x)$ where $r(x)$ is a fixed irreducible polynomial of degree n over \mathcal{F}_{p^n} .

Examples Computations over \mathcal{F}_{2^5}

$$10011 + 01110 = (1 + 0)(0 + 1)(0 + 1)(1 + 1)(1 + 0) = 11101$$

$$10011 \times 01110 = 01001 \text{ since } (x^4 + x + 1) \times (x^3 + x^2 + x) \bmod (x^5 + x^2 + 1) = x^3 + 1$$

Factoring q-1

The only efficient way we know to finding a primitive element in fields \mathcal{F}_q is when the factorization of $q-1$ is already known. In general, it may be difficult to factor $q-1$. However, if we are after a large field with a random number of elements, Eric Bach has devised an efficient probabilistic algorithm to generate random integers of a given size with known factorization. Recently, Adam Kalai has invented a somewhat slower algorithm that is much simpler. Suppose we randomly select r with its factorization using Bach's or Kalai's algorithm. We may check whether $r+1$ is a prime (power). In this case, a finite field of $r+1$ elements is obtained and a primitive element may be computed efficiently. In this course, we will limit our focus to the so-called Sophie-Germain primes:

primes p such that $q = \frac{p-1}{2}$ is also a prime.

7 Lecture7 and 8

A definition of modern cryptography: "the mathematical study of techniques for securing digital information, transactions, and distributed computations against adversarial attacks".

Private-key encryption scheme:

1. **Key-generation algorithm Gen:** probabilistic algorithm that outputs a key k . (chosen according to distribution determined by the scheme)
 2. **Encryption algorithm Enc:** inputs a key k and a plaintext message m and outputs a ciphertext c . ($Enc_k(m)$ = encryption of m using key k)
 3. **Decryption algorithm Dec:** inputs a key k and a ciphertext c and outputs a plaintext m . ($Dec_k(c)$ = decryption of c using the key k .)
- **Key space (\mathcal{K}):** set of all possible keys output by the key generation algorithm.

- Almost always, Gen simply chooses a key uniformly at random from \mathcal{K}
- **Plaintext (or message) space (\mathcal{M}):** set of all “legal” messages (i.e., those supported by the encryption algorithm).
- Since any ciphertext is obtained by encrypting some plaintext under some key, the sets \mathcal{K} and \mathcal{M} together define a set of all possible ciphertexts denoted by \mathcal{C} .
- An encryption scheme is fully defined by specifying the three algorithms (Gen, Enc, Dec) and the plaintext space \mathcal{M} .
- **Correctness** requirement of any encryption scheme is that for all k output by Gen and for all $m \in \mathcal{M}$, it holds that

$$Dec_k(Enc_k(m)) = m$$

- In words, decrypting a ciphertext (using the appropriate key) yields the original message that was encrypted.
- First, Gen is run to obtain a key that the parties share.
- When one party wants to send a plaintext m to the other
 - he computes $c \leftarrow Enc_k(m)$
 - he sends the resulting ciphertext c over the public channel to the other party.
- Upon receiving c , the other party computes $m := Dec_k(c)$ to recover the original plaintext.

Kerckhoffs’ Principle: One of the most important of these principles (now known simply as Kerckhoffs’ principle) is : ”The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience”. In other words, the encryption scheme itself should not be kept secret, and so only the key should constitute the secret information.

Three primary arguments in favor of Kerckhoffs.

1. Much easier for the parties to maintain secrecy of a short key than to maintain secrecy of an algorithm. Easier to share a short (say, 100-bit) string and store this string

securely than it is to share and securely store a program that is much larger. Furthermore, details of an algorithm can be leaked (by insider?) or learned through reverse engineering.

2. In case the key is exposed, it will be much easier for the honest parties to change the key than to replace the algorithm being used. It is good security practice to refresh a key frequently even when it has not been exposed.
3. In case many pairs of people (say, within a company) need to encrypt their communication, it will be significantly easier for all parties to use the same algorithm/program, but different keys, than for everyone to use a different program.

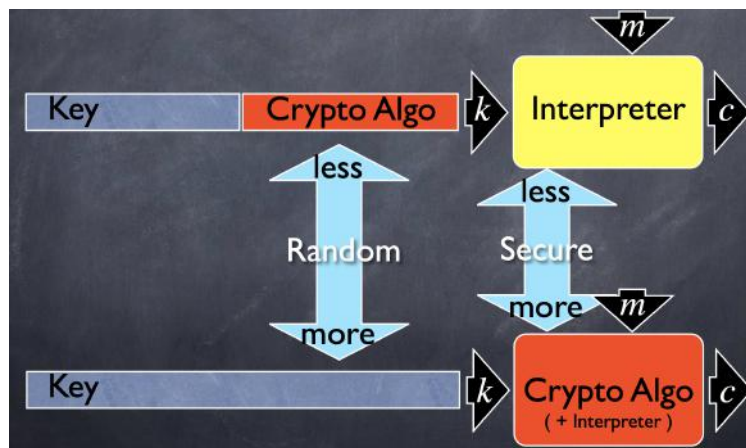


Figure 19

Today, Kerckhoffs' principle is that "all algorithms should be public".

1. Published designs undergo public scrutiny and are therefore likely to be stronger. Many years of experience have demonstrated that it is very difficult to construct good encryption schemes. Our confidence in the security of a scheme is much higher if it has been extensively studied and no weaknesses have been found.
2. It is better for security flaws, if they exist, to be revealed by "ethical hackers" (leading, hopefully, to the system being fixed) rather than having these flaws be known only to malicious parties.
3. If the security of the system relies on the secrecy of the algorithm, then reverse engineering of the code (or leakage by industrial espionage) poses a serious threat to

security. This is in contrast to the secret key which is not part of the code, and so is not vulnerable to reverse engineering.

- As simple and obvious as it may sound, the principle of open cryptographic design is ignored over and over again with disastrous results.
- Very dangerous to use a proprietary algorithm (i.e., a non-standardized algorithm that was designed in secret by someone), and only publicly tried and tested algorithms should be used.
- Enough good algorithms that are standardized but not patented, so that there is no reason whatsoever today to use something else.
- **Ciphertext-only attack:** This is the most basic type of attack and refers to the scenario where the adversary just observes a ciphertext (or multiple ciphertexts) and attempts to determine the underlying plaintext (or plaintexts).
- **Known-plaintext attack:** The adversary learns one or more pairs of plaintexts/ciphertexts encrypted under the same key. The aim is to determine the plaintext that was encrypted in some other ciphertext.
- **Chosen-plaintext attack:** The adversary has the ability to obtain the encryption of plaintexts of its choice. It then attempts to determine the plaintext that was encrypted in some other ciphertext.
- **Chosen-ciphertext attack:** The adversary is even given the capability to obtain the decryption of ciphertexts of its choice. The adversary's aim, once again, is to determine the plaintext that was encrypted in some other ciphertext.

Sufficient Key Space Principle: "Any secure encryption scheme must have a key space that is not vulnerable to exhaustive search". Necessary condition for security but not a sufficient one.

History of Ciphers:

- **Ceaser cipher:** Shift each character with fixed length, like $A \rightarrow D$, $B \rightarrow E$ etc. No security at all.

- **Shift cipher:** Introduce a secret key to represent the length we need to shift. However, the key space is not large enough.
- **Mono-alphabetic substitution:** Substitute a character with another character based on pre-determined permutations. The key space has size of $26! \approx 2^{88}$. However, we can analyze the ciphertext and recover the key plaintext by frequency analysis.
- **Vigenere cipher:** Poly-alphabetic shift cipher. Attacks on mono-alphabetic ciphers can be thwarted by mapping different instances of the same plaintext character to different ciphertext characters. This has the effect of “smoothing out” the probability distribution of characters in the ciphertext. The Vigenère cipher works by applying multiple shift ciphers in sequence. A short, secret word is chosen as the key, and then the plaintext is encrypted by “adding” each plaintext character to the next character of the key (as in the shift cipher), wrapping around in the key when necessary.

- Plaintext: tellhimaboutme
- Key: cafecafecafeca
- Ciphertext: VEQPJIREDOZXOE

If the key is a sufficiently-long word (chosen at random), then cracking this cipher seems to be a daunting task. Indeed, it was considered by many to be an unbreakable cipher. If the length of the key is known, then the task is relatively easy. Specifically, say the length of the key is t (this is sometimes called the period). Then the ciphertext can be divided into t parts where each part can be viewed as being encrypted using a single instance of the shift cipher. For every set of ciphertext characters relating to a given key, it is possible to tabulate the frequency of each ciphertext character and then check which of the 26 possible shifts yields the “right” distribution. This can be done separately for each key, the attack can be carried out very quickly; all that is required is to build t frequency tables and compare them to the real distribution.

Kasiski/Babbage’s Methods:

If the key-length is t , then the ciphertext characters $c_1, c_{1+t}, c_{1+2t}, \dots$ are encrypted using the same shift. The frequencies of the characters in this sequence are expected to be identical to the character frequencies of standard English text except in some shifted order.

Index of Coincidence Method:

Let q_i denote the frequency of the i -th English letter observed in the sequence $c_1, c_{1+t}, c_{1+2t}, \dots$. If the shift used here is k_1 , we expect q_{i-k_1} to be roughly equal to p_i for all i , where p_i

is the reference frequency of the i -th letter in English. This means that the sequence p_0, \dots, p_{25} is just the sequence q_0, \dots, q_{25} shifted by k_1 places. As a consequence, we expect that

$$\sum_{i=0}^{25} q_i^2 = \sum_{i=0}^{25} p_i^2 \approx 0.065$$

For $\tau = 1, 2, \dots$ look at the sequence of ciphertext characters $c_1, c_{1+\tau}, c_{1+2\tau}, \dots$ and tabulate q_0, \dots, q_{25} for this sequence. Then compute

$$S_\tau = \sum_{i=0}^{25} q_i^2$$

When $t = \tau$ we expect to see $S_\tau \approx 0.065$. Otherwise, for $t \neq \tau$, we will obtain $S_\tau \approx \sum_{i=0}^{25} (\frac{1}{26})^2 \approx 0.038$, which is sufficiently different from 0.065 for this technique to work.

A large ciphertext is needed for determining the period if Kasiski's method is used. Statistics are needed for t different parts of the ciphertext, and the frequency table of a message converges to the average as its length grows (and so the ciphertext needs to be approximately t times longer than in the case of the shift cipher).

Enigma Machine:

- Base 26 wheel turning mechanism
- Effectively 26^3 keys
- Keys were to be changed every week
- A key can never be mapped to itself
- Improper key usage by the German army effectively reduces the keyspace (reusing keys, using easy keys, etc.)
- Messages often repeated the same characters in the beginning or end

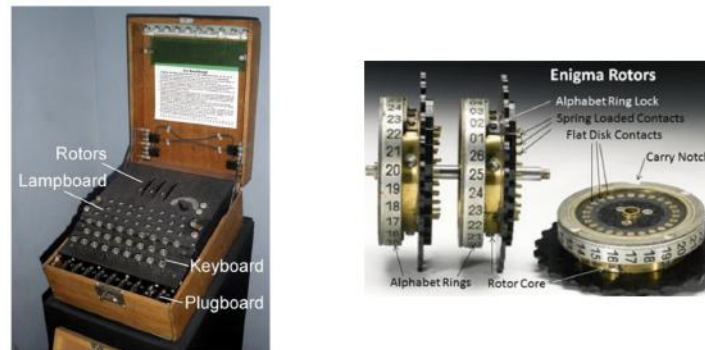


Figure 20

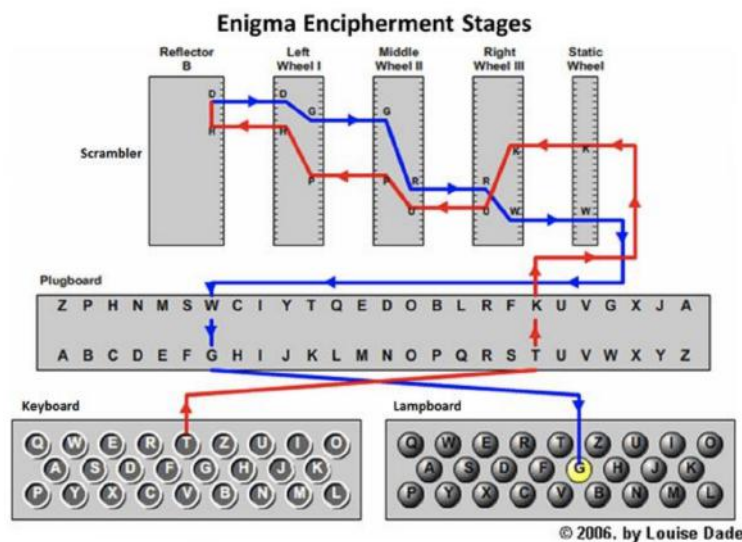


Figure 21

8 Lecture9

The Basic Principles of Modern Cryptography

1. the first step in solving any cryptographic problem is the formulation of a rigorous and precise definition of security.
2. when the security of a cryptographic construction relies on an unproven assumption, this assumption must be precisely stated. Furthermore, the assumption should be as minimal as possible.

3. cryptographic constructions should be accompanied by a rigorous proof of security with respect to a definition formulated according to Principle 1, and relative to an assumption (if any) stated as in Principle 2.

Formulation of Exact Definitions

- **Importance for design:**

- enables us to better direct our design efforts
- evaluate the quality of what we build
- define what is needed first rather than to come up with a post facto definition
- achieves more than is needed (less efficient)

- **Importance for usage:** encryption scheme within some larger system

- suffices for our application?
- define the security we desire in our system
- look for a scheme satisfying this definition
- a weaker notion of security may suffice

- **Importance for study:** how can we compare encryption schemes?

- efficiency
- level of security
- trade-off

Secure Encryption: an encryption scheme is secure if no adversary can effectively compute any function of the plaintext given the ciphertext.

There are two distinct issues that must be explicitly addressed :

- what is considered to be a break
- what is assumed regarding the power of the adversary

Any definition of security will take the following general form: A cryptographic scheme for a given task is secure if no adversary of a specified power can achieve a specified break.

Reliance on Precise Assumptions

1. Validation of the assumption:

- Assumptions are statements that are not proven but are rather conjectured to be true.
- In order to strengthen our belief in some assumption, it is necessary for the assumption to be studied.
- The more the assumption is examined and tested without being refuted, the more confident we are that the assumption is true.
- Study of an assumption can provide positive evidence of its validity by showing that it is implied by another, widely believed, assumption.
- If the assumption being relied upon is not precisely stated and presented, it cannot be studied and (potentially) refuted.

2. Comparison of schemes:

- Presented with two schemes that can both be proven to satisfy some definition but each with respect to a different assumption, assuming both schemes are equally efficient, which scheme should be preferred?
- If the assumption on which one scheme is based is weaker than the assumption on which the second scheme is based (i.e., the second assumption implies the first), then the first scheme is to be preferred.
- If the assumptions used by the two schemes are incomparable, then the general rule is to prefer the scheme that is based on the better-studied assumption, or the assumption that is simpler.

3. **Facilitation of proofs of security:** Modern cryptographic constructions are presented together with proofs of security. If the security of the scheme cannot be proven unconditionally and must rely on some assumption, then a proof that “a construction is secure if an assumption is true” can only be provided if there is a precise statement of what the assumption is.

One observation is that it is always possible to just assume that a construction itself is secure. Of course, this is not accepted practice in cryptography for a number of reasons :

- An assumption that has been tested over the years is preferable to a new assumption that is introduced just to prove a given construction secure.
- There is a general preference for assumptions that are simpler to state, since they are easier to study and to refute. For example, an assumption of the type that some mathematical problem is hard to solve is simpler to study and work with than an assumption that an encryption scheme satisfies a complex definition.
- These low-level assumptions can typically be shared amongst a number of constructions. If a specific instantiation of the assumption turns out to be false, it can simply be replaced by a different instantiation of that assumption.

The reductionist approach: Given a theorem of the form “Given that Assumption X is true, Construction Y is secure according to the given definition”, a proof typically shows how to reduce the problem given by Assumption X to the problem of breaking Construction Y. More to the point, the proof will typically show (via a constructive argument) how any adversary breaking Construction Y can be used as a sub-routine to violate Assumption X.

The distribution over \mathcal{K} is simply the one that is defined by running Gen. We let $Pr[K = k]$ denote the probability that the key output by Gen is equal to k . We let $Pr[M = m]$ denote the probability that the message is equal to m . The distributions over \mathcal{K} and \mathcal{M} are independent. This is the case because the key is chosen and fixed (i.e., shared by the communicating parties) before the message (and its distribution) is known. The distribution over \mathcal{K} is fixed by the encryption scheme itself (since it is defined by Gen). The distribution over \mathcal{M} may vary depending on the parties who are using the cipher. For $c \in \mathcal{C}$, we write $Pr[C = c]$ to denote the probability that the ciphertext C be c . Given the encryption algorithm Enc, the distribution over \mathcal{C} is fully determined by the distributions over \mathcal{K} and \mathcal{M} (and the randomness of the encryption algorithm in case it is a probabilistic algorithm).

Perfect Secrecy: An encryption scheme (Gen, Enc, Dec) over a message space \mathcal{M} is perfectly secret if for every probability distribution over \mathcal{M} , every message $m \in \mathcal{M}$, and every ciphertext $c \in \mathcal{C}$ for which $Pr[C = c] > 0$:

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

Perfect Indistinguishability: An encryption scheme (Gen, Enc, Dec) over a message space \mathcal{M} is perfectly secret if and only if for every probability distribution over \mathcal{M} , every message $m, m' \in \mathcal{M}$, and every ciphertext $c \in \mathcal{C}$,

$$\Pr[\text{Enc}_k(m) = c] = \Pr[\text{Enc}_k(m') = c]$$

Perfect Indistinguishability Proof: We show that if the stated condition holds, then the scheme is perfectly secret. Fix a distribution over \mathcal{M} , a message m , and a ciphertext c for which $\Pr[C = c] > 0$. If $\Pr[M = m] = 0$ then we trivially have

$$\Pr[M = m|C = c] = 0 = \Pr[M = m]$$

So assume $\Pr[M = m] > 0$.

Notice first that $\Pr[C = c|M = m] = \Pr[\text{Enc}_k(M) = c|M = m] = \Pr[\text{Enc}_k(m) = c]$, where the first equality is by definition of the random variable C , and the second is because we condition on the event that M is m .

Set $\delta_c = \Pr[\text{Enc}_k(m) = c] = \Pr[C = c|M = m]$.

If the condition of the lemma holds, then for every $m' \in \mathcal{M}$, we have

$$\Pr[\text{Enc}_k(m') = c] = \Pr[C = c|M = m'] = \delta_c$$

Using Bayes' Theorem, we thus have

$$\begin{aligned} \Pr[M = m|C = c] &= \frac{\Pr[C = c|M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{\Pr[C = c|M = m] \cdot \Pr[M = m]}{\sum_{m' \in \mathcal{M}} \Pr[C = c|M = m'] \cdot \Pr[M = m']} = \frac{\delta_c \cdot \Pr[M = m]}{\sum_{m' \in \mathcal{M}} \delta_c \cdot \Pr[M = m']} \\ &= \frac{\Pr[M = m]}{\sum_{m' \in \mathcal{M}} \Pr[M = m']} = \Pr[M = m] \end{aligned}$$

where the summation is over $m' \in \mathcal{M}$ with $\Pr[M = m'] > 0$.

We conclude that for every $m \in \mathcal{M}$ and $c \in \mathcal{C}$ for which $\Pr[C = c] > 0$, it holds that

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

and so the scheme is perfectly secret.

Adversarial Indistinguishability

- The experiment is defined for any encryption scheme $\Pi = (Gen, Enc, Dec)$ over message space \mathcal{M} and for any adversary A .
- We let $PrivK_{A,\Pi}^{eav}$ denote an execution of the experiment for a given Π and A . The experiment is defined as follows:

1. Adversary sends $m_0, m_1 \in \mathcal{M}$ to the challenger.
2. Challenger generates a key $k \leftarrow Gen$
3. Challenger tosses a random bit $b \leftarrow \{0, 1\}$
4. Challenger sends back $c \leftarrow Enc_k(m_b)$
5. Adversary outputs b' and challenger outputs b
6. The result of the experiment is defined to be 1 if and only $b = b'$, and 0 otherwise.

We write $PrivK_{A,\Pi}^{eav} = 1$ if the output is 1 and in this case we say that A succeeded.

- An encryption scheme $\Pi = (Gen, Enc, Dec)$ over message space \mathcal{M} is perfectly secret if for every adversary A it holds that

$$Pr[PrivK_{A,\Pi}^{eav} = 1] = \frac{1}{2}$$

Let $\Pi = (Gen, Enc, Dec)$ be an encryption scheme over message space \mathcal{M} . Then Π is perfectly secret with respect to Definition of **Perfect Secrecy** if and only if it is perfectly secret with respect to **Adversarial Indistinguishability**.

One-Time Pad

1. Fix an integer $l > 0$. Then the message space \mathcal{M} , key space \mathcal{K} , and ciphertext space \mathcal{C} are all $\{0, 1\}^l$.
2. The key-generation algorithm Gen works by uniformly choosing a string from $\mathcal{K} = \{0, 1\}^l$
3. Let $a \oplus b$ denote the bitwise exclusive-or (XOR) of binary strings a and b . Encryption Enc : given a key $k \in \{0, 1\}^l$ and a message $m \in \{0, 1\}^l$. output $c = k \oplus m$.

4. Decryption Dec: given a key $k \in \{0, 1\}^l$ and a ciphertext $c \in \{0, 1\}^l$, output $m = k \oplus c$.

We note that $Dec_k(Enc_k(m)) = k \oplus k \oplus m = m$ (a correct encryption scheme). Intuitively, the one-time pad is perfectly secret because given a ciphertext c , there is no way to know which plaintext m it comes from. In order to see why this is true, notice that for every possible m there exists a key k such that $c = Enc_k(m)$; namely, take $k = m \oplus c$. Each key is chosen with uniform probability (and hidden from the adversary) and so no key is more likely than any other.

Theorem: The one-time pad encryption scheme is perfectly-secret.

Proof: Fix some distribution over \mathcal{M} and fix arbitrary $m \in \mathcal{M}$ and $c \in \mathcal{C}$. The key observation is that for the one-time pad,

$$\begin{aligned} Pr[C = c | M = m] &= Pr[M \oplus K = c | M = m] \\ &= Pr[m \oplus K = c] = Pr[K = m \oplus c] = \frac{1}{2^l} \end{aligned}$$

Now fix any probability distribution over \mathcal{M} . For every $c \in \mathcal{C}$, we have

$$\begin{aligned} Pr[C = c] &= \sum_{m \in \mathcal{M}} Pr[C = c | M = m] \cdot Pr[M = m] \\ &= \sum_{m \in \mathcal{M}} Pr[M = m] / 2^l = \frac{1}{2^l} \end{aligned}$$

where the summation is over $m \in \mathcal{M}$, $Pr[M = m] > 0$.

Using Bayes' Theorem, we thus have

$$\begin{aligned} Pr[M = m | C = c] &= \frac{Pr[C = c | M = m] \cdot Pr[M = m]}{Pr[C = c]} \\ &= \frac{1/2^l \cdot Pr[M = m]}{1/2^l} = Pr[M = m] \end{aligned}$$

We conclude that the one-time pad is perfectly secret.

Drawbacks of OTP:

- the key is required to be as long as the message.
- a long key must be securely stored,

- limits applicability of the scheme if we send very long messages
- As the name indicates — One-time pad is only “secure” if keys are used once and re-sampled independently.
- In particular, say two messages m, m' are encrypted using the same key k . An adversary who obtains $c = m \oplus k$ and $c' = m' \oplus k$ can compute $c \oplus c' = (m \oplus k) \oplus (m' \oplus k) = m \oplus m'$ and thus learn the exclusive-or of the two messages.
- If the messages correspond to English-language text, then given the exclusive-or of two sufficiently-long messages, it has been shown to be possible to perform frequency analysis and recover the messages.

Limitations of Perfect Secrecy:

- We prove that any perfectly-secret encryption scheme must have a key space that is at least as large as the message space.
- If the key space consists of fixed-length keys, and the message space consists of all messages of some fixed length, this implies that the key must be at least as long as the message.
- The other limitation regarding the fact that the key can only be used once is also inherent.

Theorem: Let $\Pi = (Gen, Enc, Dec)$ be a perfectly-secret encryption scheme over a message space \mathcal{M} , and let \mathcal{K} be the key space as determined by Gen . Then $|\mathcal{K}| \geq |\mathcal{M}|$.

Proof: We show that if $|\mathcal{K}| < |\mathcal{M}|$ then the scheme is not perfectly secret. Assume for a contradiction that $|\mathcal{K}| < |\mathcal{M}|$. Consider the uniform distribution over \mathcal{M} and let $c \in \mathcal{C}$ be a ciphertext that occurs with nonzero probability. Let $\mathcal{M}(c)$ be the set of all possible messages which are possible decryptions of c ; that is

$$\mathcal{M}(c) = \{m \mid m = Dec_k(c) \text{ for some } k \in \mathcal{K}\}$$

Clearly $|\mathcal{M}(c)| \leq |\mathcal{K}|$ since for each message $m \in \mathcal{M}(c)$ we can identify at least one key $k \in \mathcal{K}$ for which $m = Dec_k(c)$ (Don't forget that Dec is deterministic).

Under the assumption that $|\mathcal{K}| < |\mathcal{M}|$, this means that there is some $m' \in \mathcal{M}$ such that $m' \notin \mathcal{M}(c)$ but then

$$\Pr[M = m' | C = c] = 0 \neq \Pr[M = m']$$

and so the scheme is not perfectly secret.

Shannon's Theorem: Let $\Pi = (Gen, Enc, Dec)$ be an encryption scheme over a message space \mathcal{M} for which $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$. The scheme is perfectly secret if and only if:

1. Every $k \in \mathcal{K}$ is chosen with equal probability $= \frac{1}{|\mathcal{K}|}$ by Gen.
2. For every $m \in \mathcal{M}$ and every $c \in \mathcal{C}$, there exists a unique $k \in \mathcal{K}$ such that $Enc_k(m)$ outputs c .

Uses of Shannon's Theorem

- Theorem 2.11 is of interest in its own right in that it essentially gives a complete characterization of perfectly secret encryption schemes.
- In addition, since items (1) and (2) have nothing to do with the probability distribution over the set of plaintexts \mathcal{M} , the theorem implies that if there exists an encryption scheme that provides perfect secrecy for a specific probability distribution over \mathcal{M} then it actually provides perfect secrecy in general (i.e., for all probability distributions over \mathcal{M}).
- Shannon's theorem is extremely useful for proving whether a scheme is or is not perfectly secret.
- Item (1) is easy to confirm and item (2) can be demonstrated (or contradicted) without analyzing any probabilities.
- The perfect secrecy of the one-time pad is trivial to prove using Shannon's theorem.
- Theorem 2.11 only holds if $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$ and so one must be careful to apply it only in this case.

9 Lecture10

A computational Approach to Cryptography

- Modern encryption schemes have the property that they can be broken given enough time.
- Do not satisfy Definition of perfect secrecy, but for all practical purposes, the following level of security suffices.
- Under certain assumptions, the amount of computation needed to break these encryption schemes would take more than many lifetimes to carry out even using the fastest available supercomputers.

The Basic Idea of Computational Security:

- Kerckhofs actually spelled out six principles, the following of which is very relevant to our discussion here: A [cipher] must be practically, if not mathematically, indecipherable.
- The computational approach incorporates two relaxations of the notion of perfect security:
 1. Security is only preserved against efficient adversaries that run in a feasible amount of time
 2. Adversaries can potentially succeed with some very small probability.

The concrete approach quantifies the security of a given cryptographic scheme by explicitly bounding the maximum success probability of any adversary running for at most some fixed amount of time. That is, let t, ϵ be positive constants with $\epsilon \leq 1$. A concrete definition of security: A scheme is (t, ϵ) – *secure* if every adversary running for time at most t succeeds in breaking the scheme with probability at most ϵ .

Modern private-key encryption schemes are generally assumed to give almost optimal security in the following sense: When the key has length n , an adversary running in time t can succeed in breaking the scheme with probability at most $\frac{ct}{2^n}$ for some fixed constant c .

The asymptotic approach: We require that honest parties run in polynomial time, Concerned with achieving security against polynomial-time adversaries. Adversarial strategies that require a superpolynomial amount of time are not considered realistic threats (and so are essentially ignored). We equate the notion of “small probability of success” with success probabilities smaller than any inverse polynomial in n , meaning that for every constant c the adversary’s success probability is smaller than n^{-c} for all large enough values of n . A function that grows slower than any inverse polynomial is called negligible. A definition of asymptotic security thus takes the following form: A scheme is secure if every **Probabilistic Polynomial Time (PPT)** adversary succeeds in breaking the scheme with only negligible probability.

Necessity of the Relaxations:

- Assume we have an encryption scheme where the size of the key space \mathcal{K} is much smaller than the size of the message space \mathcal{M} .
- Two attacks, lying at opposite extremes, apply regardless of how the encryption scheme is constructed:

1. Brute-Force Search

- Given a ciphertext c , an adversary can decrypt c using all keys $k \in \mathcal{K}$.
- This gives a list of all possible messages to which c can possibly correspond.
- Since this list cannot contain all of \mathcal{M} (because $|\mathcal{K}| < |\mathcal{M}|$), this leaks some information about the message that was encrypted.
- Moreover, say the adversary carries out a knownplaintext attack and learns that ciphertexts c_1, \dots, c_l correspond to the messages m_1, \dots, m_l respectively.
- The adversary can again try decrypting each of these ciphertexts with all possible keys until it finds a key k for which $Dec_k(c_i) = m_i$ for all i .
- This key will be unique with high probability, in which case the adversary has found the key that the honest parties are using.
- Subsequent usage of this key will therefore be insecure.
- The type of attack succeeds with probability essentially 1 in time linear in $|\mathcal{K}|$.

2. Random Attack

- Consider again the case where the adversary learns that c_1, \dots, c_l corresponds to m_1, \dots, m_l .

- The adversary can guess a key $k \in \mathcal{K}$ at random and check to see whether $Dec_k(c_i) = m_i$ for all i .
- If so, we again expect that with high probability k is the key that the honest parties are using.
- Here the adversary runs in essentially constant time and succeeds with non-zero (although very small) probability of roughly $\frac{1}{|\mathcal{K}|}$.

We define efficient computation as that which can be carried out in Probabilistic Polynomial Time (abbreviated PPT). An algorithm A is said to run in polynomial time if there exists a polynomial $p(\cdot)$ such that, for every input $x \in \{0,1\}^*$, the computation of $A(x)$ terminates within at most $p(|x|)$ steps. A probabilistic algorithm is one that has the capability of “tossing coins”; This is a metaphorical way of saying that the algorithm has access to a source of randomness that yields unbiased random bits that are each independently equal to 1 with probability $1/2$ and to 0 with probability $1/2$.

Negligible Success: A function f is negligible if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that

$$f(n) < \frac{1}{p(n)}$$

Let $negl_1$ and $negl_2$ be negligible functions of an integer n . Then

1. The function $negl_3(n) = negl_1(n) + negl_2(n)$ is also negligible.
2. For any positive polynomial p , the function $negl_4(n) = p(n) \cdot negl_1(n)$ is also negligible.

The general framework of any security definition will be: A scheme is secure if for every PPT adversary A carrying out an attack of some specified type, the probability that A succeeds in this attack (where success is also well-defined) is negligible.

In order to see this in more detail, we will use the full definition of “negligible” in the above statement: A scheme is secure if for every PPT adversary A carrying out an attack of some specified type, and for every polynomial $p(\cdot)$, there exists an integer N such that the probability that A succeeds is less than $1/p(n)$ for every $n > N$.

Defining Computationally Secure Encryption: A private-key encryption scheme is a tuple of probabilistic polynomial-time algorithms (Gen, Enc, Dec) such that:

1. The key-generation algorithm Gen takes as input the security parameter 1^n and outputs a key k ; we write this as $k \leftarrow \text{Gen}(1^n)$ (thus emphasizing the fact that Gen is a randomized algorithm). We will assume without loss of generality that any key $k \leftarrow \text{Gen}(1^n)$ satisfies $|k| \leq n$.
 2. The encryption algorithm Enc takes as input a key k and a plaintext message $m \in \{0, 1\}^*$, and outputs a ciphertext c . Since Enc may be randomized, we write $c \leftarrow \text{Enc}_k(m)$.
 3. The decryption algorithm Dec takes as input a key k and a ciphertext c , and outputs a message m . We assume that Dec is deterministic, and so write this as $m = \text{Dec}_k(c)$.
- It is required that for every n , every key k output by $\text{Gen}(1^n)$, and every $m \in \{0, 1\}^*$, it holds that

$$\text{Dec}_k(\text{Enc}_k(m)) = m$$

- If $(\text{Gen}, \text{Enc}, \text{Dec})$ is such that for k output by $\text{Gen}(1^n)$, algorithm $\text{Enc}_k(m)$ is only defined for $m \in \{0, 1\}^{l(n)}$, then we say that $(\text{Gen}, \text{Enc}, \text{Dec})$ is a fixed-length private-key encryption scheme for messages of length $l(n)$.

Computationally Secret ($\text{PrivK}_{A,\Pi}^{\text{eav}}(n)$):

1. The adversary A is given input 1^n , and outputs a pair of messages $m_0, m_1 \in \mathcal{M}$ of the same length.
2. A key k is generated by running $\text{Gen}(1^n)$, and a random bit $b \leftarrow \{0, 1\}$ is chosen. A (challenge) ciphertext $c \leftarrow \text{Enc}_k(m_b)$ is computed and given to A .
3. A outputs a bit b' .
4. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. (If $\text{PrivK}_{A,\Pi}^{\text{eav}}(n) = 1$, we say that A succeeded.)

If Π is a fixed-length scheme for messages of length $l(n)$, the previous experiment is modified by requiring $m_0, m_1 \in \{0, 1\}^{l(n)}$

Definition of Computationally-Secure Encryption: A private-key encryption scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ has indistinguishable encryptions in the presence of an eavesdropper

if for all PPT adversaries A there exists a negligible function $\text{negl}(n)$ such that

$$\Pr[\text{PrivK}_{A,\Pi}^{\text{eav}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

where the probability is taken over the random coins used by A , as well as the random coins used in the experiment (for choosing the key, the random bit b , and any random coins used in the encryption process).

Semantic Security: A private-key encryption scheme has indistinguishable encryptions in the presence of an eavesdropper if and only if it is semantically secure in the presence of an eavesdropper.

10 Lecture11

Pseudorandomness actually refers to a distribution over strings, and when we say that a distribution D over strings of length l is pseudorandom this means that D is indistinguishable from the uniform distribution over strings of length l . Strictly speaking, since we are in an asymptotic setting we actually need to speak of the pseudorandomness of a sequence of distributions $D = \{D_n\}_{n \in \mathbb{N}}$, where distribution D_n is associated with security parameter n . We ignore this point in our current discussion. More precisely, it is infeasible for any PPT algorithm to tell whether it is given a string sampled according to D or an l -bit string chosen uniformly at random.

Pseudorandom Generators: A pseudorandom generator is a deterministic algorithm that receives a short truly random seed and stretches it into a long string that is pseudorandom. Stated differently, a pseudorandom generator uses a small amount of true randomness in order to generate a large amount of pseudorandomness. In the definition that follows, we set n to be the length of the seed that is input to the generator and $l(n)$ to be the output length. The generator is only interesting if $l(n) > n$.

Definition of Pseudorandom Generators: Let $l(\cdot)$ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any input $s \in \{0,1\}^n$, algorithm G outputs a string of length $l(n)$. The function l is called the expansion factor of G . We say that G is a pseudorandom generator if the following two conditions hold:

1. **Expansion:** For every n it holds that $l(n) > n$.

2. **Pseudorandomness:** For all PPT distinguishers D , there exists a negligible function $\text{negl}(n)$ such that

$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1]| \leq \text{negl}(n)$$

where r is chosen uniformly at random from $\{0, 1\}^{l(n)}$, the seed s is chosen uniformly at random from $\{0, 1\}^n$, and the probabilities are taken over the random coins used by D and the choice of r and s . $D(x) = 1$ means D votes x is pseudo-random. $D(x) = 0$ means D votes x is truly random.

Pseudorandom Generators Discussion:

- It is trivial to distinguish between a random string and a pseudorandom string given an unlimited amount of time.
- Upon input some string w , distinguisher D outputs 1 if and only if there exists a string $s \in \{0, 1\}^n$ such that $G(s) = w$.

$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1]| \geq 1 - 2^{n-l(n)}$$

- The seed for a pseudorandom generator must be chosen uniformly at random, and be kept entirely secret from the distinguisher.
- Another important point, evident from the above discussion of brute-force attacks, is that s must be long enough so that no “efficient algorithm” has time to traverse all possible seeds.
- Technically, this is taken care of by the fact that all algorithms are assumed to run in polynomial time and thus cannot search through all 2^n possible seeds when n is large enough.
- Unfortunately, we do not know how to unequivocally prove the existence of pseudorandom generators.
- We believe that pseudorandom generators exist, and this belief is based on the fact that they can be constructed under the rather weak assumption that one-way functions exist.
- In practice, various constructions believed to act as pseudorandom generators are known.

Stream Ciphers: Formally, we view a stream cipher as a pair of deterministic algorithms (Init, GetBits) where:

- Init takes as input a seed s and an optional initialization vector IV , and outputs an initial state st_0 .
- GetBits takes as input state information st_{i-1} , and outputs a bit y_i and updated state st_i . (In practice, y_i is a block of several bits; we treat y as a single bit here for generality and simplicity.)

ALGORITHM 3.16

Constructing G_ℓ from (Init, GetBits)

Input: Seed s and optional initialization vector IV

Output: y_1, \dots, y_ℓ

$st_0 := \text{Init}(s, IV)$

for $i = 1$ **to** ℓ :

$(y_i, st_i) := \text{GetBits}(st_{i-1})$

return y_1, \dots, y_ℓ

Figure 22

Stream Ciphers Example

- $st_0 = \text{Init}(s, IV) = (s \cdot IV)^2 \bmod N$
- $(y_i, st_i) = \text{Getbits}(st_{i-1}) = (\text{lsb}(st_{i-1}), st_{i-1}^2 \bmod N)$

Example

$N = 2643463649$, $s = 1234756234$, $IV = 172653$, $st_0 = 2225733751$

- 19000213261
- 604632310
- 4584019701
- 14946635510

- 13510593931
- 18457811501

$N = 2643463649$, $s = 1234756234$, $IV = 111333$, $st_0 = 1184429618$

- 25144954420
- 8944953460
- 15445839900
- 23569907040
- 5665114830
- 4098569361

A stream cipher is secure:

- In the basic sense if it takes no IV and for any polynomial l with $l(n) > n$, the function G_l is a pseudorandom generator with expansion factor l .
- One possible security notion for stream ciphers that use an IV is discussed in Section 3.6.1.

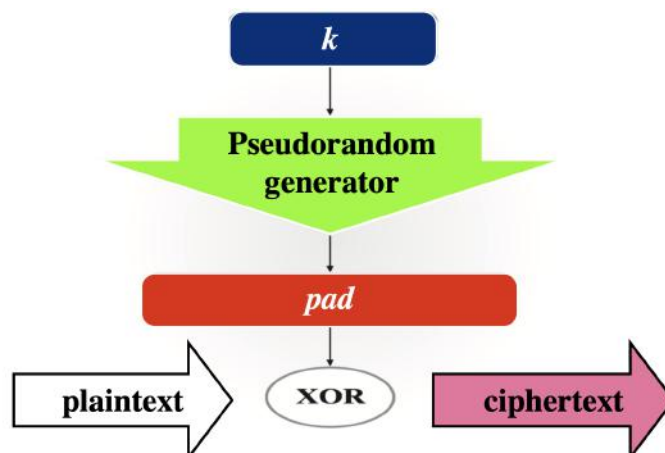


Figure 23

CONSTRUCTION 3.17

Let G be a pseudorandom generator with expansion factor ℓ . Define a private-key encryption scheme for messages of length ℓ as follows:

- **Gen:** on input 1^n , choose uniform $k \in \{0, 1\}^n$ and output it as the key.
- **Enc:** on input a key $k \in \{0, 1\}^n$ and a message $m \in \{0, 1\}^{\ell(n)}$, output the ciphertext

$$c := G(k) \oplus m.$$
- **Dec:** on input a key $k \in \{0, 1\}^n$ and a ciphertext $c \in \{0, 1\}^{\ell(n)}$, output the message

$$m := G(k) \oplus c.$$

A private-key encryption scheme based on any pseudorandom generator.

Figure 24

Theorem: If G is a pseudorandom generator, then Construction 3.17 is a fixedlength private-key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper.

The point of the construction, of course, is that the ℓ -bit string $G(k)$ can be much longer than the key k .

Security for Multiple Encryptions: $\text{PrivK}_{A,\Pi}^{\text{Mult}}(n)$

1. The adversary A is given input 1^n , and outputs a pair of vectors of messages $M_0 = (m_{10}, \dots, m_{t0})$ and $M_1 = (m_{11}, \dots, m_{t1})$ with $|m_{i0}| = |m_{i1}|$ for $1 \leq i \leq t$
2. A key k is generated by running $\text{Gen}(1^n)$, and a random bit $b \leftarrow \{0, 1\}$ is chosen. For all i , the ciphertext $c_i \leftarrow \text{Enc}_k(m_{ib})$ is computed and the vector of ciphertexts $C = (c_1, \dots, c_t)$ is given to A .
3. A outputs a bit b' .
4. The output of the experiment is defined to be 1 if $b'=b$, and 0 otherwise.

Security for Multiple Encryptions: $\text{PrivK}_{A,\Pi}^{\text{Mult}}(n)$: A private-key encryption scheme

$\Pi = (Gen, Enc, Dec)$ has indistinguishable multiple encryptions in the presence of an eavesdropper if for all PPT adversaries A there exists a negligible function $negl(n)$ such that

$$Pr[PrivK_{A,\Pi}^{Mult}(n) = 1] \leq \frac{1}{2} + negl(n)$$

where the probability is taken over the random coins used by A , as well as the random coins used in the experiment (for choosing the key and the random bit b , as well as for the encryption itself).

Proposition: There exist private-key encryption schemes that have indistinguishable encryptions in the presence of an eavesdropper but do not have indistinguishable multiple encryptions in the presence of an eavesdropper. In the proof of this proposition we show that Construction 3.17 is not secure for multiple encryptions. The only feature of that construction used in the proof [is] that encrypting a message always yields the same ciphertext, and so we actually obtain that any deterministic scheme must be insecure for multiple encryptions.

Theorem: Let $\Pi = (Gen, Enc, Dec)$ be an encryption scheme for which Enc is a deterministic function of the key and the message. Then Π does not have indistinguishable multiple encryptions in the presence of an eavesdropper.

11 Lecture12

Chosen Plaintexts Attack: This is formalized by allowing A to interact freely with an encryption oracle, viewed as a “black-box” that encrypts messages of A ’s choice using the secret key k . We denote by $A^{O(\cdot)}$ the computation of A given access to an oracle O . We denote the computation of A with access to an encryption oracle that uses key k by $A^{Enc_k(\cdot)}$.

Experiment of CPA $PrivK_{A,\Pi}^{CPA}(n)$:

1. A key k is generated by running $Gen(1^n)$.
2. The adversary A is given input 1^n and oracle access to Enc_k , and outputs a pair of messages m_0, m_1 of the same length.
3. A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow Enc_k(m_b)$ is created and given to A . We call c the challenge ciphertext.

4. The adversary A continues to have oracle access to Enc_k , and outputs a bit b' .
5. The output of the experiment is defined to be 1 if $b'=b$, and 0 otherwise. (When $PrivK_{A,\Pi}^{CPA}(n) = 1$, we say that A succeeded.)

Indistinguishable encryptions under CPA: A private-key encryption scheme $\Pi = (Gen, Enc, Dec)$ has indistinguishable encryptions under a chosen-plaintext attack (or is CPA-secure) if for all probabilistic polynomial-time adversaries A there exists a negligible function $negl$ such that

$$Pr[PrivK_{A,\Pi}^{CPA}(n) = 1] \leq \frac{1}{2} + negl(n)$$

where the probability is over the random coins used by A , as well as the random coins used in the experiment.

Any scheme that has indistinguishable encryptions under a chosen-plaintext attack clearly also has indistinguishable encryptions in the presence of an eavesdropper. This holds because $PrivK_{A,\Pi}^{eav}(n)$ is a special case of $PrivK_{A,\Pi}^{CPA}(n)$ where the adversary doesn't use its oracle at all.

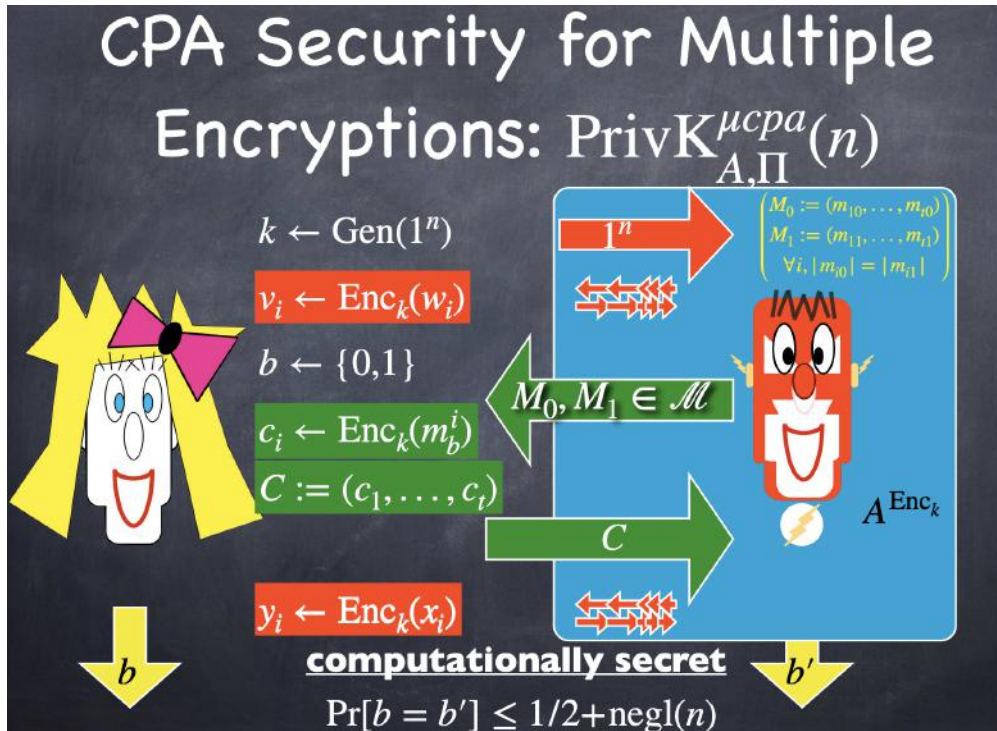


Figure 25: $PrivK_{A,\Pi}^{\mu CPA}(n)$

Experiment of $PrivK_{A,\Pi}^{L\vee R-cpa}(n)$:

1. A key k is generated by running $Gen(1^n)$.
2. A random bit $b \leftarrow \{0, 1\}$ is chosen.
3. The adversary A is given the security parameter 1^n and oracle access to $LVR_{k,b}(m_0, m_1) = Enc_k(m_b)$, and uses it for many pairs of inputs chosen adaptively.
4. The adversary outputs a bit b' , his guess of b .
5. The output of the experiment is defined to be 1 if $b'=b$, and 0 otherwise. (When $PrivK_{A,\Pi}^{LVR-cpa}(n) = 1$, we say that A succeeded.)

CPA security for adaptive multiple encryptions $PrivK_{A,\Pi}^{LVR-cpa}(n)$: A private-key encryption scheme $\Pi = (Gen, Enc, Dec)$ has indistinguishable multiple encryptions under a chosen-plaintext attack (or is CPA-secure) if for all probabilistic polynomial-time adversaries A there exists a negligible function $negl$ such that

$$Pr[PrivK_{A,\Pi}^{LVR-cpa}(n) = 1] \leq \frac{1}{2} + negl(n)$$

where the probability is over the random coins used by A , as well as the random coins used in the experiment.

Any private-key encryption scheme that has indistinguishable encryptions under a chosen-plaintext attack also has indistinguishable multiple encryptions under a chosen-plaintext attack.

Pseudorandom Functions: Let $F : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be an efficient, length-preserving, keyed function. We say that F is a Pseudorandom Function if for all PPT distinguishers D , there exists a negligible function $negl$ such that:

$$|Pr[D^{F_k(\cdot)}(1^n) = 1] - Pr[D^{f(\cdot)}(1^n) = 1]| \leq negl(n)$$

where $k \leftarrow \{0, 1\}^n$ is chosen uniformly at random and f is chosen uniformly at random from the set of functions mapping n -bit strings to n -bit string.

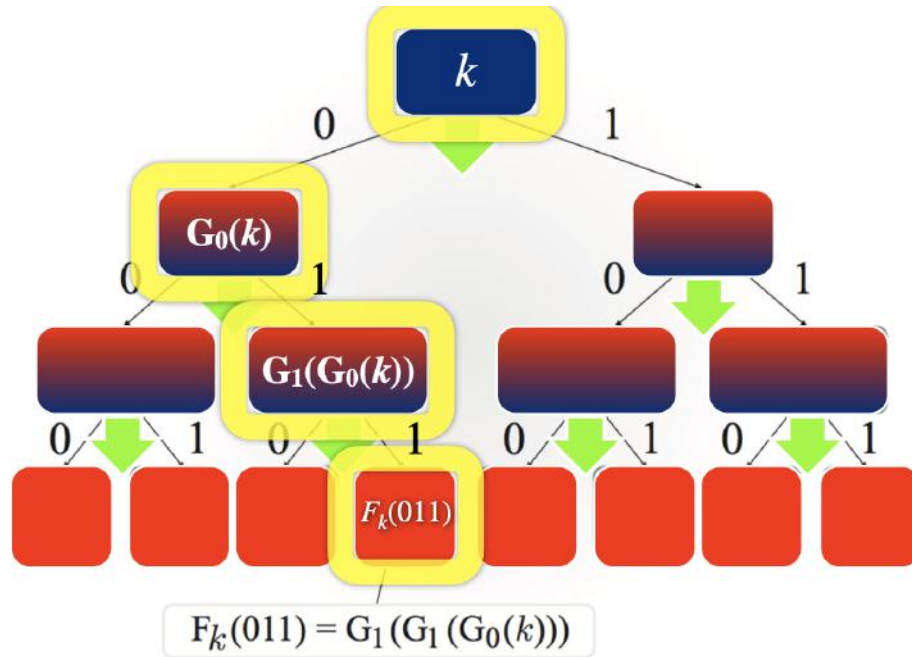


Figure 26: Construct Pseudorandom Function from Pseudorandom Generators

CONSTRUCTION 8.20

Let G be a pseudorandom generator with expansion factor $\ell(n) = 2n$, and define G_0, G_1 as in the text. For $k \in \{0, 1\}^n$, define the function $F_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$ as:

$$F_k(x_1 x_2 \cdots x_n) = G_{x_n}(\cdots (G_{x_2}(G_{x_1}(k))) \cdots).$$

A pseudorandom function from a pseudorandom generator.

$$F_k(10011) = G_1(G_1(G_0(G_0(G_1(k)))))$$

Theorem: If G is a Pseudorandom Generator with expansion factor $\ell(n) = 2n$, then Construction 8.20 is a Pseudorandom Function.

Pseudorandom Permutations and Block Ciphers

- Let $F : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be an efficient, length-preserving, keyed functions.
- We call F a keyed permutation if for every k , the function $F(\cdot)$ is a bijection.
- We say that a keyed permutation is efficient if there is a PPT algorithm computing $F_k(x)$ given k and x , as well as a PPT algorithm computing $F_k^{-1}(x)$ given k and x .

- We define what it means for an efficient keyed permutation F to be a pseudorandom permutation in a manner exactly analogous to Definition of pseudorandom function.
- The only change is that we now require that F_k (for a randomly-chosen k) be indistinguishable from a randomly-chosen permutation rather than a randomly-chosen function.

Proposition: If F is a Pseudorandom Permutation then it is also a Pseudorandom Function.

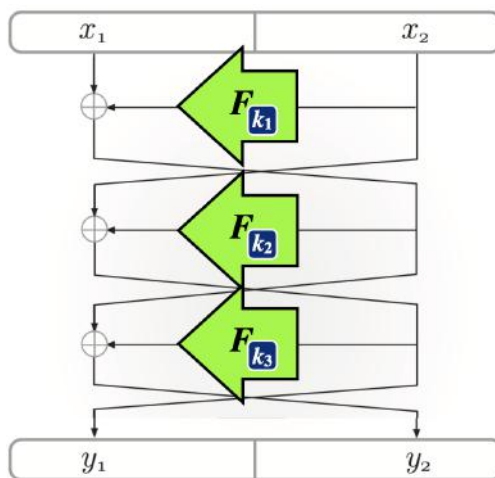


Figure 28: Pseudorandom Permutations

CONSTRUCTION 8.23*

Let F be a keyed, length-preserving function. Define the keyed permutation $F^{(3)}$ as follows:

- **Inputs:** A key $k = (k_1, k_2, k_3)$ with $|k_i| = n$, and an input $x \in \{0, 1\}^{2n}$ parsed as (L_0, R_0) with $|L_0| = |R_0| = n$.
- **Computation:**
 1. Compute $L_1 := R_0$ and $R_1 := L_0 \oplus F_{k_1}(R_0)$.
 2. Compute $L_2 := R_1$ and $R_2 := L_1 \oplus F_{k_2}(R_1)$.
 3. Compute $L_3 := R_2$ and $R_3 := L_2 \oplus F_{k_3}(R_2)$.
 4. Output (L_3, R_3) .

Figure 29: Pseudorandom Permutations

Theorem: If F is a length-preserving Pseudorandom Function, then $F^{(3)}$ is a Pseudorandom Permutation that maps $2n$ -bit strings to $2n$ -bit strings (and uses a key of length $3n$).

12 Lecture13

If F is an efficient pseudorandom permutation then cryptographic schemes based on F might require honest parties to compute the inverse F_k^{-1} in addition to the permutation F_k itself. This potentially introduces new security concerns that are not covered by the fact that F is pseudorandom. We may need to impose the stronger requirement that F_k be indistinguishable from a random permutation even if the distinguisher is additionally given oracle access to the inverse of the permutation. If F has this property, we call it a **Strong Pseudorandom Permutation**.

Pseudorandom Permutations and Block Ciphers: Let $F : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be an efficient, keyed permutation. We say that F is a strong pseudorandom function if for all PPT distinguishers D , there exists a negligible function negl such that:

$$|Pr[D^{F_k(\cdot), F_k^{-1}(\cdot)}(1^n) = 1] - Pr[D^{f(\cdot), f^{-1}(\cdot)}(1^n) = 1]| \leq \text{negl}(n)$$

where $k \leftarrow \{0, 1\}^n$ is chosen uniformly at random and f is chosen uniformly at random from the set of permutations on n -bit strings.

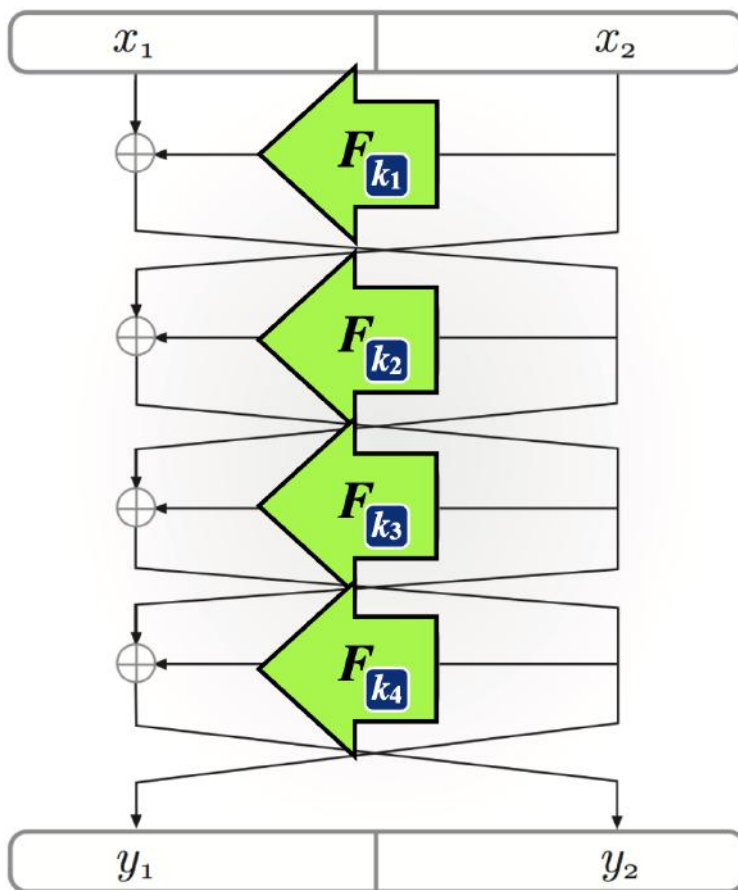


Figure 30: Strong Pseudorandom Permutations

CONSTRUCTION 8.23

Let F be a keyed, length-preserving function. Define the keyed permutation $F^{(4)}$ as follows:

- **Inputs:** A key $k = (k_1, k_2, k_3, k_4)$ with $|k_i| = n$, and an input $x \in \{0, 1\}^{2n}$ parsed as (L_0, R_0) with $|L_0| = |R_0| = n$.
- **Computation:**
 1. Compute $L_1 := R_0$ and $R_1 := L_0 \oplus F_{k_1}(R_0)$.
 2. Compute $L_2 := R_1$ and $R_2 := L_1 \oplus F_{k_2}(R_1)$.
 3. Compute $L_3 := R_2$ and $R_3 := L_2 \oplus F_{k_3}(R_2)$.
 4. Compute $L_4 := R_3$ and $R_4 := L_3 \oplus F_{k_4}(R_3)$.
 5. Output (L_4, R_4) .

A strong pseudorandom permutation from any pseudorandom function.

Figure 31: Strong Pseudorandom Permutations

Theorem: If F is a length-preserving Pseudorandom Function, then $F^{(4)}$ is a Strong Pseudorandom Permutation that maps $2n$ -bits strings to $2n$ -bit strings (and uses a key of length $4n$).

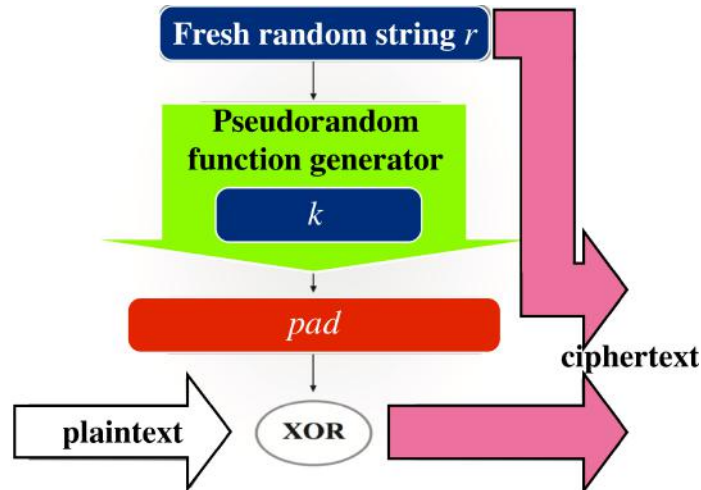


Figure 32: CPA-Secure Encryption from Pseudorandom Functions

Given a scheme that is based on a pseudorandom function, a general way of analyzing the scheme is to first prove its security under the assumption that a truly random function is used instead. Next, the security of the original scheme is derived by proving that if an adversary can break the scheme when a pseudorandom function is used, then it must implicitly be distinguishing the function from random.

CONSTRUCTION 3.30

Let F be a pseudorandom function. Define a private-key encryption scheme for messages of length n as follows:

- **Gen:** on input 1^n , choose uniform $k \in \{0, 1\}^n$ and output it.
- **Enc:** on input a key $k \in \{0, 1\}^n$ and a message $m \in \{0, 1\}^n$, choose uniform $r \in \{0, 1\}^n$ and output the ciphertext

$$c := \langle r, F_k(r) \oplus m \rangle.$$

- **Dec:** on input a key $k \in \{0, 1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the plaintext message

$$m := F_k(r) \oplus s.$$

Figure 33

Theorem: If F is a pseudorandom function, then Construction 3.30 is a fixed-length private-

key encryption scheme for messages of length n that has indistinguishable encryptions under CPA.

Efficiency of Construction 3.30

- Construction 3.30 has the drawback that the length of the ciphertext is (at least) double the length of the plaintext.
- This is because each block of size n is encrypted using an n -bit random string which must be included as part of the ciphertext.
- In Section 3.6.2 we will show how the ciphertext length can be significantly reduced.

Block Cipher Modes of Operation

- Note that arbitrary-length messages can be unambiguously padded to a total length that is a multiple of any desired block size by appending a 1 followed by sufficiently-many 0s.
- We will therefore just assume that the length of the plaintext message is an exact multiple of the block size.
- Throughout this section, we will refer to a pseudorandom permutation/block cipher F with block length n , and will consider the encryption of messages consisting of l blocks each of length n .

1. Electronic Code Book mode (ECB)

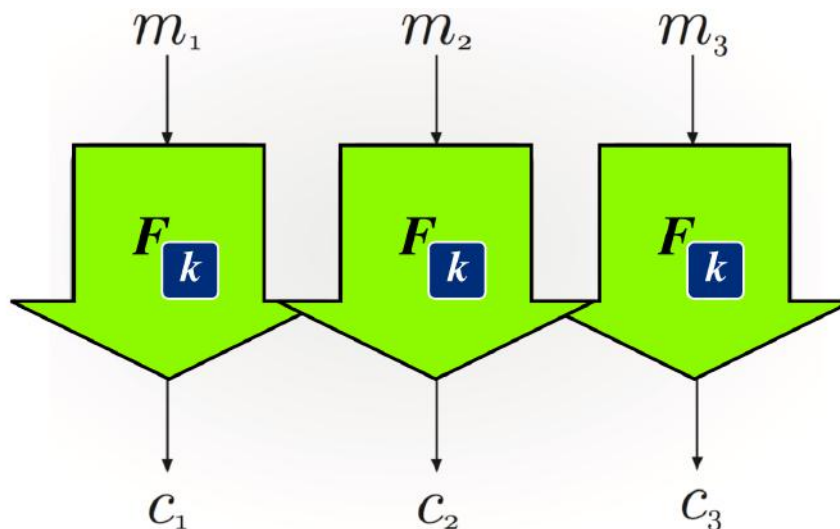


Figure 34: ECB

- This is the most naive mode of operation possible.
- Given a plaintext message $m = m_1, m_2, \dots, m_l$ the ciphertext is obtained by “encrypting” each block separately.
- “Encryption” here means a direct application of the pseudorandom permutation to the plaintext block:

$$c = F_k(m_1), \dots, F_k(m_l)$$

- Decryption is carried in the obvious way, using the fact that F_k^{-1} is efficiently computable.
- This encryption process is deterministic and therefore this mode of operation cannot be CPA-secure.
- Even worse, ECB-mode encryption does not have indistinguishable (multiple) encryptions in the presence of an eavesdropper. (This is due to the fact that if the same block is repeated twice in the plaintext, this can be detected as a repeating block in the ciphertext.)
- It is easy to distinguish an encryption of a plaintext that consists of two identical blocks from an encryption of a plaintext that consists of two different blocks.

2. Cipher Block Chaining mode (CBC)

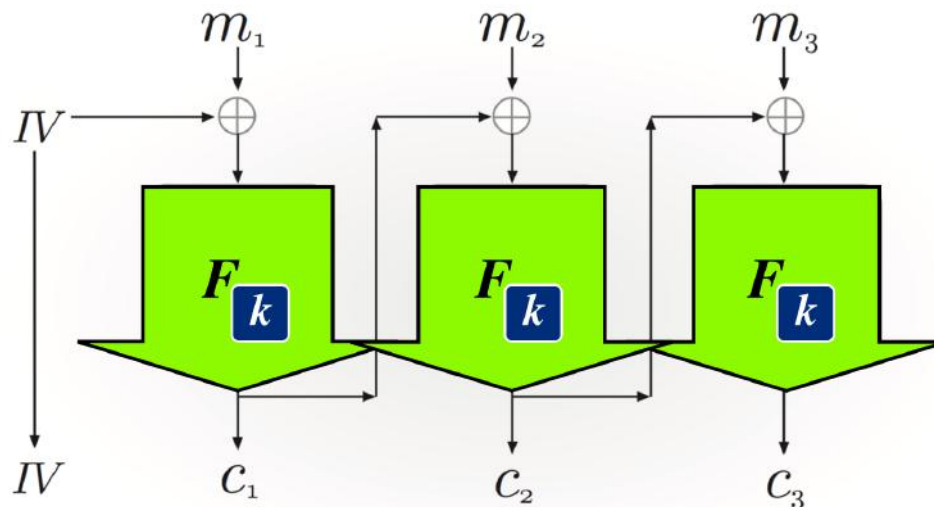


Figure 35: CBC

- In this mode, a random initial vector (IV) of length n is first chosen.
- Each of the remaining ciphertext blocks is generated by applying the pseudorandom permutation to the XOR of the current plaintext block and the previous ciphertext block.
- Enc: Set $c_0 = IV$ and then for $i=1$ to l , set $c_i = F_k(c_{i-1} \oplus m_i)$. The final ciphertext is c_0, \dots, c_l .
- We stress that the IV is sent in the clear as part of the ciphertext; this is crucial so that decryption can be carried out:
- Dec: $c_0 = IV$ and then for $i=1$ to l , set $m_i = c_{i-1} \oplus F_k^{-1}(c_i)$.
- Importantly, encryption in CBC mode is probabilistic and it has been proven that if F is a pseudorandom permutation then CBC-mode encryption is CPA-secure.
- The main drawback of this mode is that encryption must be carried out sequentially because the ciphertext block c_{i-1} is needed in order to encrypt the plaintext block m_i .
- If parallel processing is available, CBC-mode encryption may not be the most efficient choice.

3. Output Feedback mode(OFB):

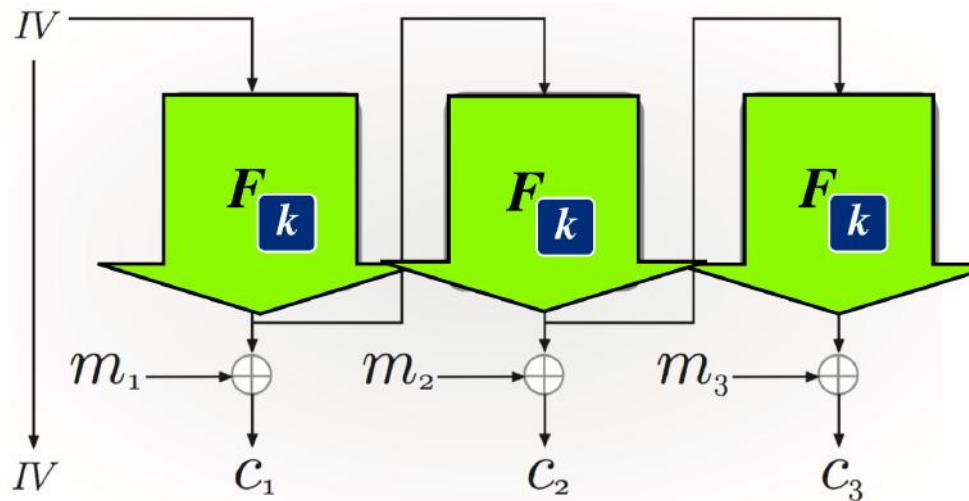


Figure 36: OFB

- Essentially, this mode is a way of using a block cipher to generate a pseudorandom stream that is then XORed with the message.
- First, a random $IV \leftarrow \{0,1\}^n$ is chosen and a stream is generated from IV (independently of the plaintext message) in the following way:
- Enc: Set $r_0 = IV$ and then, for $i=1$ to l , set $r_i = F_k(r_{i-1})$. The final ciphertext is $r_0, c_1 = r_1 \oplus m_1, \dots, c_l = r_l \oplus m_l$.
- This mode is also probabilistic, and it can be shown that it too is a CPA-secure encryption scheme if F is a pseudorandom function.
- This mode has the advantage that the bulk of the computation can be done independently of the actual message to be encrypted. Using pre-processing, encryption of the plaintext (once it is known) is incredibly fast.
- In contrast to CBC mode, here it is not required that F be invertible (in fact, it need not even be a permutation)

4. Counter Mode (CTR)

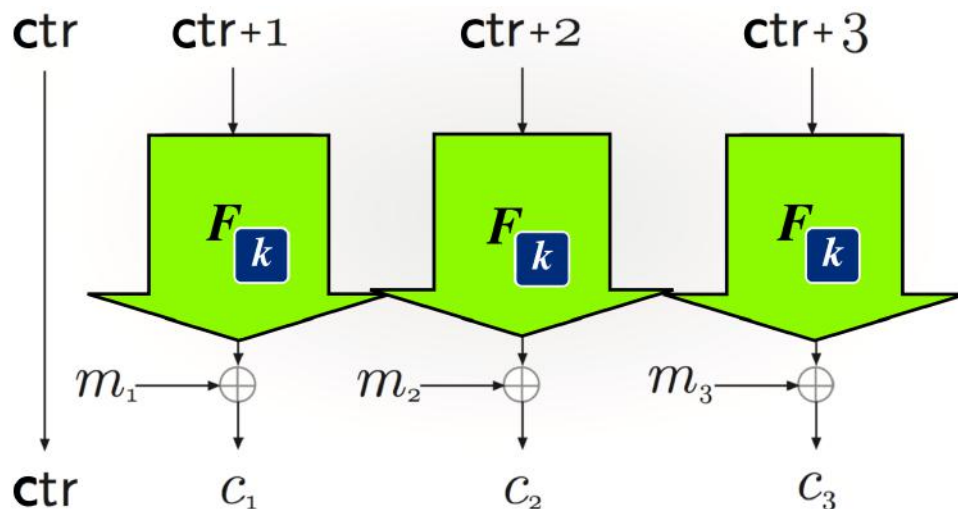


Figure 37: CTR

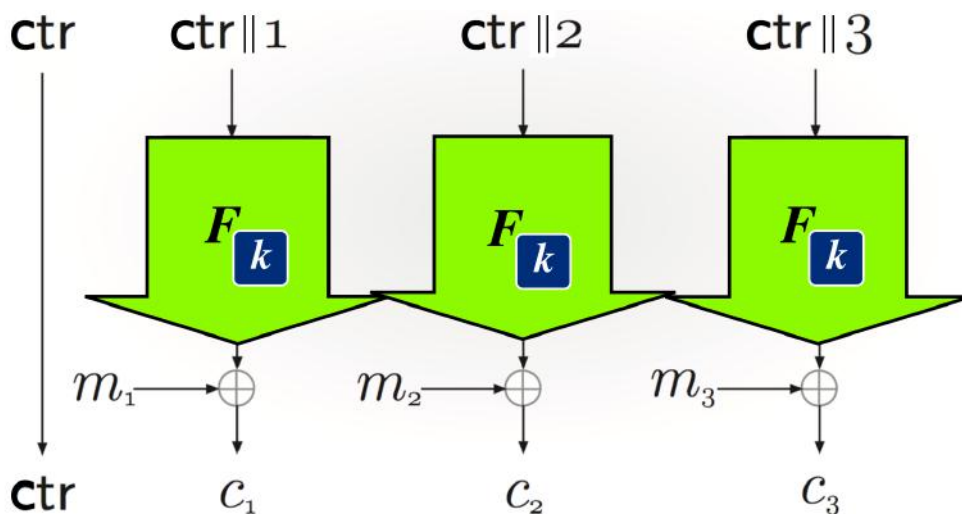


Figure 38: CTR

- There are different variants of CTR-mode encryption; we describe the randomized counter mode here.
- As with OFB, counter mode can be viewed as a way of generating a pseudorandom stream from a block cipher.
- Enc: (i) First, a random $ctr \leftarrow \{0,1\}^n$ is chosen. (ii) A stream is generated as $r_i = F_k(ctr + i)$ (where ctr and i are viewed as integers and addition is performed modulo 2^n). (iii) Finally, the i -th block is computed as $c_i = r_i \oplus m_i$, and the ctr is again sent as part of the ciphertext.

- Note once again that decryption does not require F to be invertible, or even a permutation.
- First and foremost, randomized counter mode is CPA-secure.
- Second, both encryption and decryption can be fully parallelized and, as with OFB mode, it is possible to generate the pseudorandom stream ahead of time, independently of the message.
- Finally, it is possible to encrypt and decrypt the i -th block of the ciphertext without en/decrypting anything else; this property is called random access.

Block length and security:

- Most of the above modes use a random IV.
- The IV has the effect of randomizing the encryption process, and ensures that (with high probability) the block cipher is always evaluated on a new input that was never used before.
- This is important because, if an input to the block cipher is used more than once then security can be violated. (E.g., in the case of counter mode, the same pseudorandom string will be XORed with two different plaintext blocks.)
- Interestingly, this shows that it is not only the key length of a block cipher that is important in evaluating its security, but also its block length. For example, say we use a block cipher with a 64-bit block length.
- The IV is then a uniform n -bit string, and we expect an IV to repeat after encrypting about $2^{n/2}$ messages. If n is too short, then even if F is secure as a pseudorandom permutation—the resulting concrete-security bound will be too weak for practical applications.
- This remains asymptotically negligible (when the block length grows as a function of the security parameter n), but security no longer holds in any practical sense in many cases.
- If $n = 64$, then after roughly $2^{32} \approx 4,300,000,000$ encryptions - or roughly 34 gigabytes of plaintext - a repeated IV is expected to occur.
- Although this may seem like a lot of data, it is smaller than the capacity of modern hard drives.

Attacks on Block Ciphers

- **Ciphertext-only attacks**, where the attacker is given only outputs $\{F_k(x_i)\}$ for some unknown inputs $\{x_i\}$.
- **Known-plaintext attacks**, where the attacker is given pairs of inputs and outputs $\{x_i, F_k(x_i)\}$
- **Chosen-plaintext attacks**, where the attack is given $\{x_i, F_k(x_i)\}$ for inputs $\{x_i\}$ chosen by the attacker.
- **Chosen-ciphertext attacks**, where the attacker is given $\{x_i, F_k(x_i)\}$ and $\{y_i, F_k^{-1}(y_i)\}$ for $\{x_i\}, \{y_i\}$ chosen by the attacker.

This is interpreted very strictly in the context of block ciphers, and a block cipher is generally only considered “good” if the best known attack has time complexity roughly equivalent to a brute-force search for the key.

The confusion-diffusion paradigm: Say we want F to have a block length of 128 bits. Define F as follows:

- The key k for F will specify 16 random permutations f_1, \dots, f_{16} that each have an 8-bit block length.
- Given an input $x \in \{0, 1\}^{128}$, we parse it as 16 consecutive bytes x_1, \dots, x_{16} and then set $F_k(x) = f_1(x_1) || \dots || f_{16}(x_{16})$.
- Following confusion, a diffusion step is introduced whereby the bits of the output are permuted or “mixed”.
- The confusion/diffusion steps — together called a round — are repeated multiple times.
- As an example, a two-round block cipher would operate as follows:
 1. First, $x' = F_k(x)$ would be computed.
 2. Then the bits of x' would be re-ordered (permuted) to give x_1 .
 3. Then $x'_1 = F_k(x_1)$ would be computed, and the bits of x'_1 would be re-ordered to give the output x_2 .

- Repeated use of confusion and diffusion ensures that any small change in the input will be mixed throughout and propagated to all the bits of the output.
- The effect is that small changes to the input have a significant effect on the output, as one would expect of a random permutation.

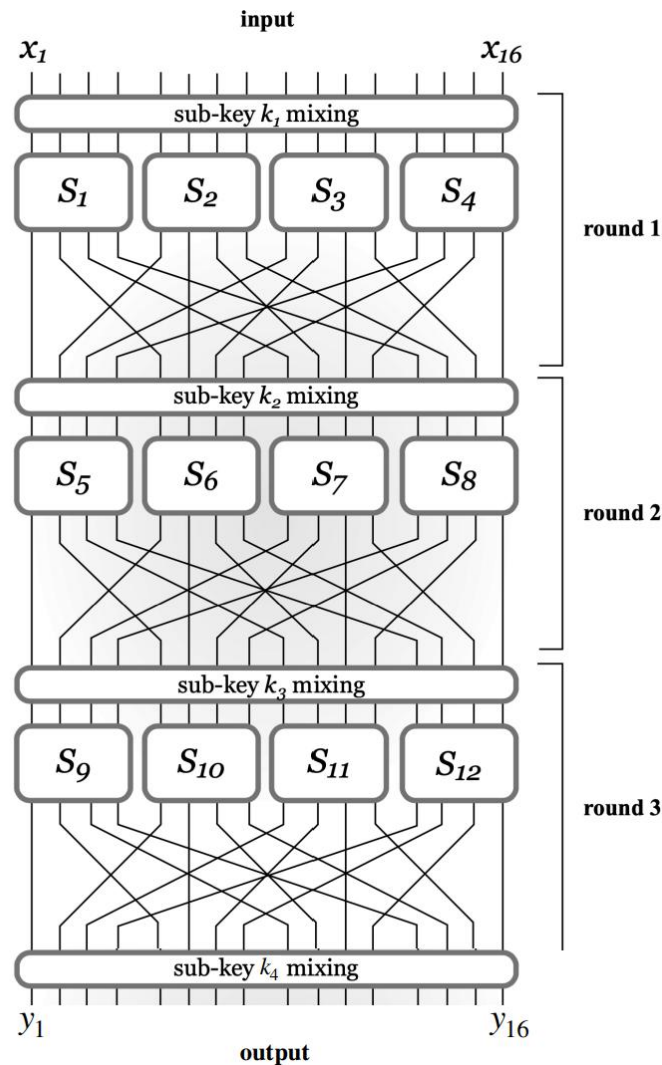


Figure 39: Substitution Permutation Network

Substitution Permutation Network

- A substitution-permutation network can be viewed as a direct implementation of the confusion-diffusion paradigm

- The main difference here is that we view the round functions $\{f_i\}$ as being fixed, and the key is used for a different purpose.
- We now refer to the $\{f_i\}$ as S-boxes
- A substitution-permutation network essentially follows the steps of the confusion-diffusion paradigm outlined earlier.
- Since the S-boxes no longer depend on the key, we need to introduce dependence in some other way.
- (In accordance with Kerckhoffs' principle, we assume that the exact structure of the S-boxes and the mixing permutations are publicly-known, with the only secret being the key.)
- There are many ways this can be done.
- We will focus here on the case where this is done by simply XORing some function of the key with the intermediate results that are fed as input to each round of the network.

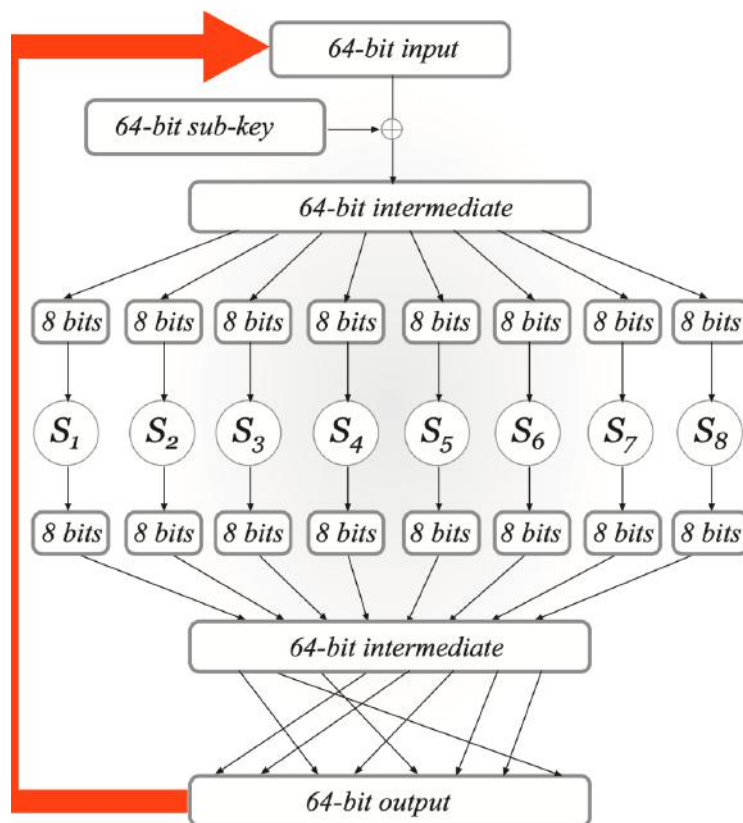


Figure 40: Substitution Permutation Network

Subs.-Perm. Network: General Structure

1. Key mixing: Set $x = x \oplus k$, where k is the current-round sub-key
 2. Substitution: Set $x = S_1(x_1) || \dots || S_8(x_8)$, where x_i is the i -th byte of x
 3. Permutation: Permute the bits of x to obtain the output of the round.
- The key to the block cipher is sometimes referred to as the master key.
 - The sub-keys that are XORed with the intermediate results in each round are derived from the master key according to a key schedule.
 - The key schedule is often very simple and may work by just taking subsets of the bits of the key, though more complex schedules can also be defined.

Design principles

1. invertibility of the S -boxes

- In a substitution-permutation network, the S-boxes must be invertible; that is, they must be one-to-one and onto functions.
- The reason for this is that otherwise the block cipher will not be a permutation.
- To see that making the S-boxes one-to-one and onto suffices, we show that when this holds it is possible to fully determine the input given the output and the key.
- Specifically, we show that every round can be inverted.
- This implies that the entire network can be inverted by working from the end back to the beginning.
- **Proposition:** Let F be a keyed function defined by a substitution-permutation network in which the S-boxes are all one-to-one and onto (permutations). Then regardless of the key schedule and the number of rounds, F_k is a permutation for any choice of k .

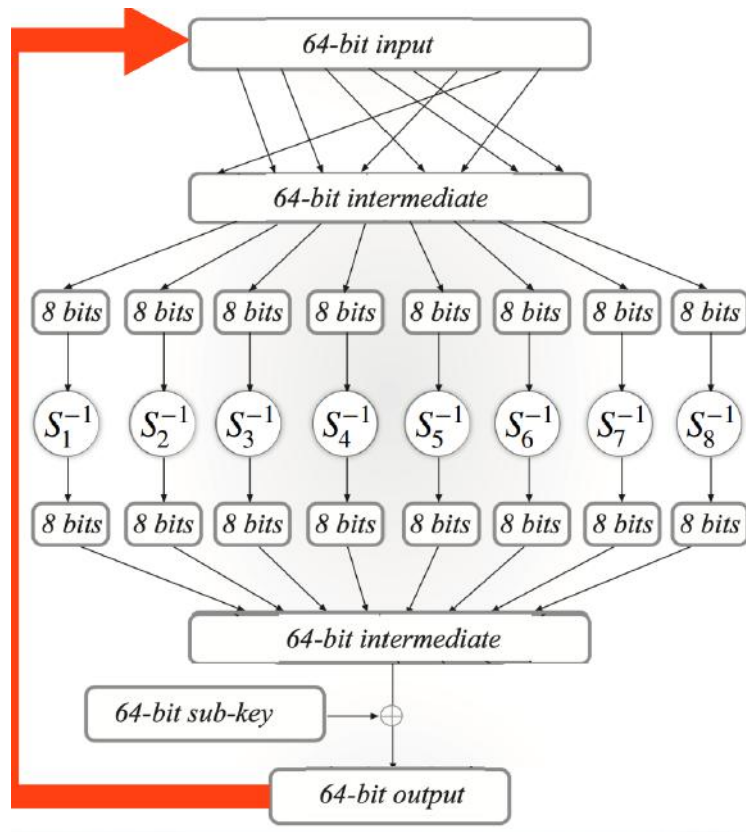


Figure 41: Invertibility of the S -boxes

2. the avalanche effect

- An important property in any block cipher is that small changes to the input must result in large changes to the output.
- Otherwise, the outputs of the block cipher on two similar inputs will not look independent
- (whereas in a random permutation, the outputs of any two unequal inputs are independently and uniformly distributed).
- Block ciphers are designed to exhibit the avalanche effect, meaning that changing a single bit of the input affects every bit of the output.
- (This does not mean that changing one bit of the input changes every bit of the output, only that it has some effect on every bit of the output.)
- It is easy to demonstrate that the avalanche effect holds in a substitution-permutation network provided that the following two properties hold (and sufficiently many rounds are used):

- (a) The S-boxes are designed so that changing a single bit of the input to an S-box changes at least two bits in the output of the S-box.
- (b) The mixing permutations are designed so that the output bits of any given S-box are spread into different S-boxes in the next round.
- Consider now what happens when the block cipher is applied to two inputs that differ by only a single bit:
 - (a) After the first round, the intermediate values differ in exactly two bit-positions. This is because XORing the current sub-key maintains the 1-bit difference in the intermediate values, and so the inputs to all the S-boxes except one are identical. In the one S-box where the inputs differ, the output of the S-box causes a 2-bit difference. The mixing permutation applied to the results changes the positions of these differences, but maintains a 2-bit difference.
 - (b) By the second property mentioned earlier, the mixing permutation applied at the end of the first round spreads the two bit-positions where the intermediate results differ into two different S-boxes in the second round. So, in the second round there are now two S-boxes that receive inputs differing by a single bit. Following the same argument as before, we see that at the end of the second round the intermediate values differ in 4 bits.

We conclude based on this example that, as a general rule, it is best to carefully design S-boxes with certain desired properties (in addition to the one discussed above) rather than choosing them blindly at random. Experience, along with many years of cryptanalytic effort, indicate that substitution-permutation networks are a good choice for constructing pseudorandom permutations as long as great care is taken in the choice of the S-boxes, the mixing permutations, and, less importantly, the key schedule. The Advanced Encryption Standard (AES), described in Section 5.5, is similar in structure to the substitution-permutation network described above, and is widely believed to be a very strong pseudorandom permutation.

13 Lecture14

Attacks on reduced round Subs-Per Network

- The adversary begins with the output value y and then inverts the mixing permutation and the S-boxes. It can do this because the specification of the permutation and the

S-boxes is public.

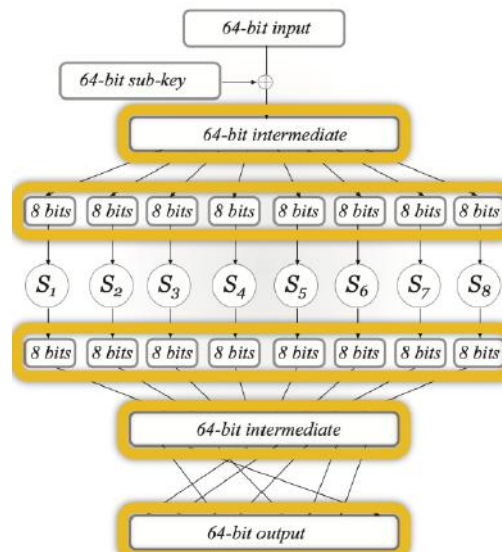


Figure 42

- The intermediate value that the adversary computes from these inversions is exactly $x \oplus k$ (assuming, without loss of generality, that the master key is used as the sub-key in the only round of the network).

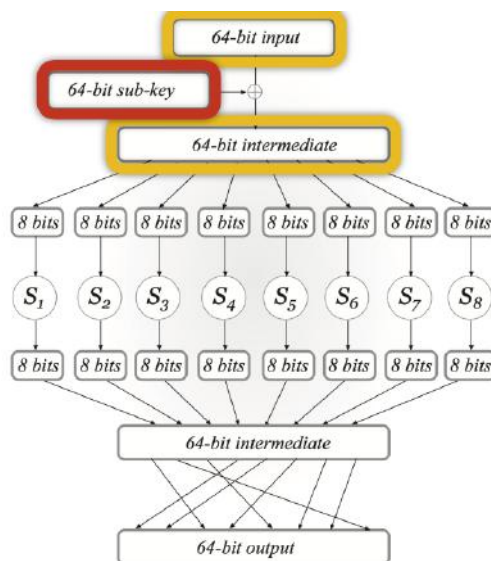


Figure 43

- Since the adversary also has the input x , it immediately derives the secret key k . This is therefore a complete break.

Feistel Networks: A Feistel network is an alternative approach for constructing a block cipher. The low-level building blocks (S-boxes, mixing permutations, and a key schedule) are the same. The difference is in the high-level design. A Feistel network is a way of constructing an invertible function from non-invertible components. The advantage of Feistel networks over substitution-permutation networks is that a Feistel network eliminates the requirement that S-boxes be invertible. This is important because a good block cipher should have “unstructured” behaviour. However, requiring that all the components of the construction be invertible inherently introduces structure. In a Feistel network, round functions need not be invertible. Round functions typically contain components like S-boxes and mixing permutations. But a Feistel network can deal with any round functions irrespective of their design.

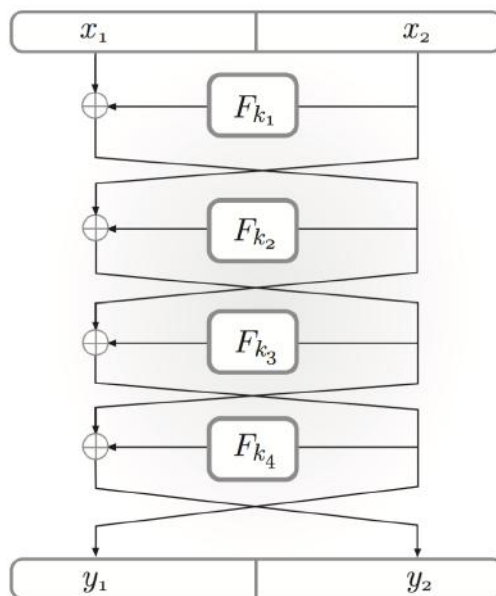


Figure 44: Feistel Networks

Operation of Feistel Network (The i -th round of a Feistel network operates as follows):

1. The input to the round is divided into two halves denoted L_{i-1} and R_{i-1} . If the block length of the cipher is n bits, then L_{i-1} and R_{i-1} each has length $n/2$. The i -th round function f_i will take an $n/2$ -bit input and produce an $n/2$ -bit output.
2. The output (L_i, R_i) of the round is given by $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f_i(R_{i-1})$. In a t -round Feistel network, the n -bit input to the network is parsed as (L_0, R_0) , and the output is the n -bit value (L_t, R_t) obtained after applying all t rounds.

- The master key k is used to derive sub-keys that are used in each round.
- The i -th round function f_i depends on the i -th subkey, denoted k_i .
- Formally, the design of a Feistel network specifies a publicly-known mangler function f'_i associated with each round i .
- This function f'_i takes as input a sub-key k_i and an $n/2$ -bit string and outputs an $n/2$ -bit string.
- When the master key is fixed the i -th round function f_i is defined via $f_i(R) = f'_i(k_i, R)$.
- A Feistel network is invertible regardless of the round functions $\{f_i\}$ (and of the mangler functions $\{f'_i\}$). Given the output L_i, R_i of the i -th round, we can compute (L_{i-1}, R_{i-1}) as follows: (i) First set $R_{i-1} = L_i$. (ii) Then compute $L_{i-1} = R_i \oplus f_i(R_{i-1})$.

Proposition: Let F be a keyed function defined by a Feistel network. Then regardless the mangler functions $\{f'_i\}$ and the number of rounds, F_k is a permutation for any choice of k .

Data Encryption Standard (DES):

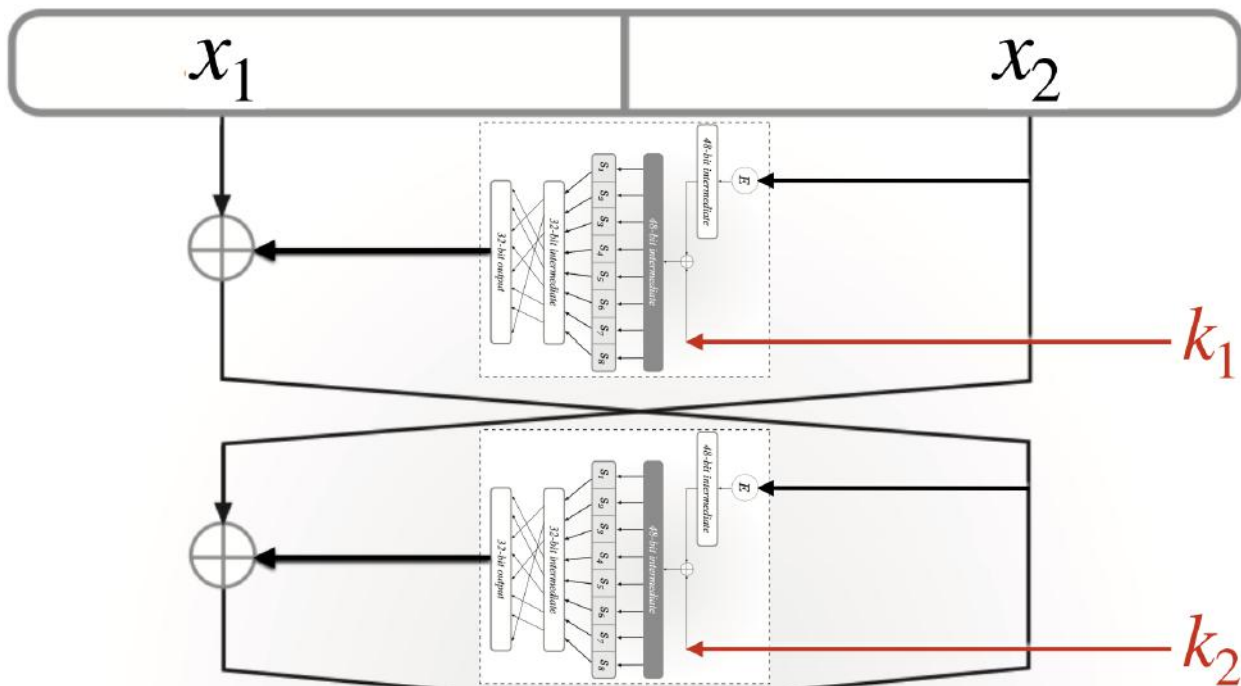


Figure 45: DES

- The DES block cipher is a 16-round Feistel network with a block length of 64 bits and a key length of 56 bits. The input/output length of a DES round function is 32 bits.
- The round functions used in each of the 16 rounds of DES are all derived from the same mangler function $f'_i = f'$. The key schedule of DES is used to derive a 48-bit sub-key k_i for each round from the 56-bit master key k . The i -th round function f_i is defined as

$$f_i(R) = f'(k_i, R)$$

The round functions are non-invertible.

- The key schedule of DES is relatively simple, with each sub-key k_i being a permuted subset of 48 bits from the master key. We will not describe the key schedule exactly. It suffices for us to note that the 56 bits of the master key are divided into two halves – a “left half ” and a “right half ” – each containing 28 bits.
- In each round, the left-most 24 bits of the sub-key are taken as some subset of the 28 bits in the left half of the master key, and the rightmost 24 bits of the sub-key are taken as some subset of the 28 bits in the right half of the master key.

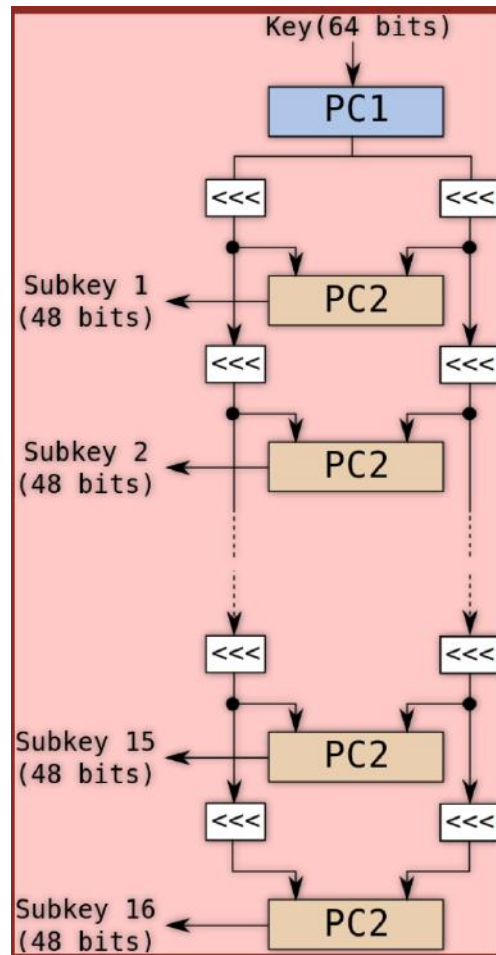
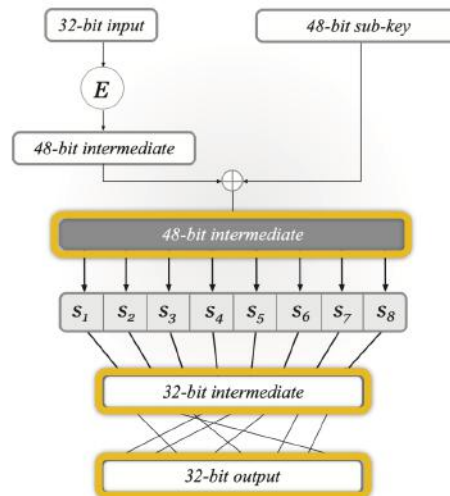


Figure 46: DES

The mangler function in DES is constructed using a paradigm we have previously analyzed: It is (essentially) just a 1-round substitution permutation network! In more detail, computation of $f'(k_i, R)$ with $k_i \in \{0, 1\}^{48}$ and $R \in \{0, 1\}^{32}$ proceeds as follows:

1. R is expanded to a 48-bit value R' . This is done by simply duplicating half the bits of R ; we denote this by $R' = E(R)$ where E represents the expansion function.
2. The expanded value R' is XORed with k_i , and the resulting value divided into 8 blocks, each of which is 6 bits long.
3. Each block is passed through a different S-box that takes a 6-bit input and yields a 4-bit output; concatenating the output from the 8 S-boxes gives a 32-bit result.
4. As the final step, a mixing permutation is applied to the bits of this result to obtain the final output of f' .

Figure 47: DES mangler function f'

S-boxes The eight S-boxes that form the “core” of f' are a crucial element of the DES construction, and were very carefully designed (reportedly, with the help of the National Security Agency). Studies of DES have shown that if small changes to the S-boxes had been introduced, or if the S-boxes had been chosen at random, DES would have been much more vulnerable to attack. Warning: seemingly arbitrary choices are not arbitrary at all, and if not made correctly may render the entire construction insecure. Recall that each S-box maps 6-bit strings to 4-bit strings. Each S-box can be viewed as a table with 4 rows and 16 columns, where each cell of the table contains a 4-bit entry. A 6-bit input can be viewed as indexing one of the $2^6 = 64$ cells of the table in the following way:

S_5		Middle 4 bits of input															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits of input	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	1001	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Figure 48: $S_5 \rightarrow S_5(011011) = 1001$

- The first and last input bits are used to choose the table row, and bits 2-5 are used to choose the column.
- The 4-bit entry at a particular cell represents the output value for the input associated with that position.

1. Each S-box is a 4-to-1 function. (each output has exactly 4 pre-images)
2. Each row in the table contains each of the 16 possible 4-bit strings exactly once. (That is, each row is a permutation of the 16 possible 4-bit strings.
3. Changing one bit of the input always changes at least two bits of the output.

The third property of the DES S-boxes described above, along with the mixing permutation that is used in the mangler function, ensure that DES exhibits a strong avalanche effect. DES has 16 rounds, and so the avalanche effect is completed early (8-th round) in the computation. This ensures that the computation of DES on similar inputs yields completely independent-looking outputs. We remark that the avalanche effect in DES is also due to a careful choice of the mixing permutation, and in fact it has been shown that a random mixing permutation would yield a weaker effect. Clearly DES with three rounds or fewer cannot be a pseudorandom function because the avalanche effect is not yet complete after only three rounds. Thus, we will be interested in demonstrating more difficult (and more damaging) key-recovery attacks which compute the key k using only a relatively small number of input/output pairs computed using that key.

Attack for single-round DES: In single-round DES, we have that $y = (L_1, R_1)$ where $L_1 = R_0$ and $R_1 = L_0 \oplus f_1(R_0)$. We therefore know an input/output pair for f_1 ; specifically, we know that $f_1(R_0) = R_1 \oplus L_0$ (all these values are known).

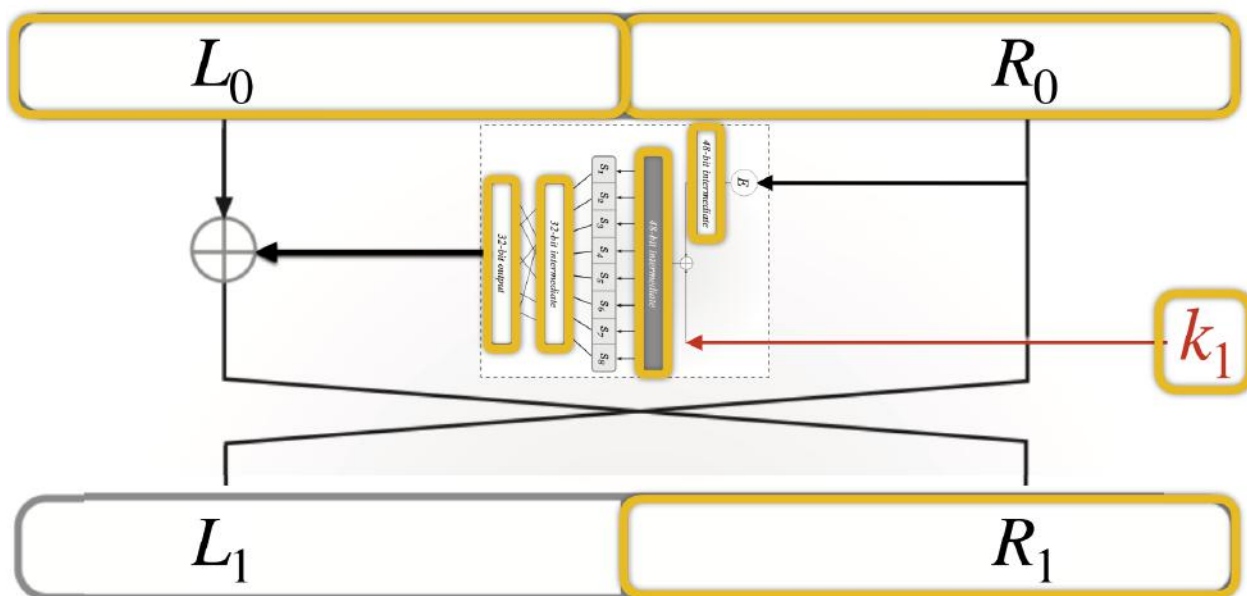


Figure 49: Single Round DES

1. By applying the inverse of the mixing permutation to the output $R_1 \oplus L_0$ we obtain the intermediate value that contains the outputs from all the S-boxes, where the first 4 bits are the output from the first S-box, the next 4 bits are the output from the second S-box, and so on. This means that we have the exact output of each S-box.
2. The input to the S-boxes is simply the XOR of $E(R_0)$ with the key k_1 used in this round. (Actually, for single-round DES, k_1 is the only key.) Since R_0 is known, we conclude that for each 6-bit portion of k_1 there are four possible values (and compute them).
 - This means we have reduced the number of possible keys k_1 from 2^{48} to $4^8 = 2^{16}$ (since there are four possibilities for each of the eight 6-bit portions of k_1)
 - This is already a small number and so we can just try all the possibilities on a different input/output pair (x', y') to find the right one (by intersecting the two lists of possible keys).
 - Recovers full key using two known plaintexts in time roughly 2^{16} .

Attack for Two-round DES: In two-round DES, the output y is equal to (L_2, R_2) where $L_1 = R_0, R_1 = L_2 = L_0 \oplus f_1(R_0), L_2 = R_1 = L_0 \oplus f_1(R_0), R_2 = L_1 \oplus f_2(R_1)$. Note that L_0, R_0, L_2, R_2 are known from the input/output pair and thus we also know $f_1(R_0) = L_0 \oplus L_2$ and $f_2(L_2) = R_0 \oplus R_2$.

This means that we know the input/output of f_1 and f_2 , and so the same method used in the attack on singleround DES can be used here to determine both k_1 and k_2 in time roughly $2 \cdot 2^{16}$. This attack works even if k_1 and k_2 are completely independent keys. In fact, the key schedule of DES ensures that many of the bits of k_1 and k_2 are equal, which can be used to further speed up the attack.

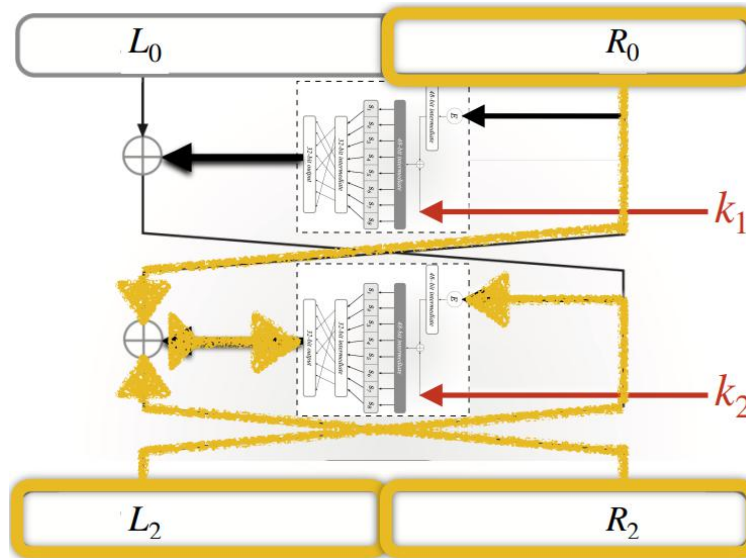


Figure 50: Two-Round DES

Attack for Three-round DES: The time complexity of the attack is roughly $2 \cdot 2^{28} + 2^{24} < 2^{30}$, and its space complexity is $2 \cdot 2^{12}$. An attack of this complexity could be carried out on a standard personal computer.

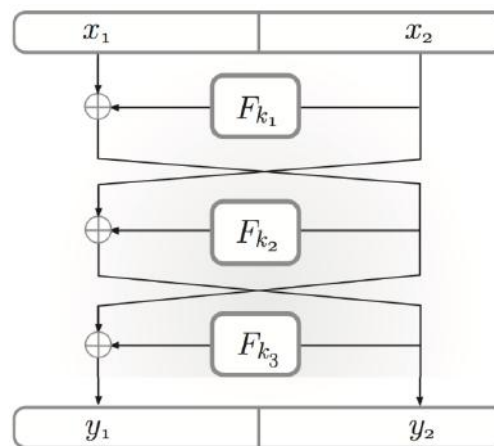


Figure 51: Three-Round DES

Differential Cryptanalysis

- Let $\Delta_x, \Delta_y \in \{0, 1\}^n$. We say that the differential (Δ_x, Δ_y) appears with probability p if for random input x and choice of key k , the probability that $F_k(x) \oplus F_k(x \oplus \Delta_x) = \Delta_y$ is p .

- It is clear that for a random function, no differential should appear with probability p much higher than 2^{-2n} .
1. Interestingly, the work of Biham and Shamir indicated that the DES S-boxes had been specifically designed to be resistant to differential cryptanalysis (to some extent), suggesting that the technique of differential cryptanalysis was known (but not publicly revealed) by the designers of DES.
 2. After Biham and Shamir announced their result, the designers of DES (represented by Don Coppersmith) claimed that they were indeed aware of differential cryptanalysis and had designed DES to thwart this type of attack (but were asked by the NSA to keep it quiet in the interests of national security).
- The only known practical weakness of DES is its relatively short key.
 - It thus makes sense to try to design a block cipher with a larger key length using “basic” DES as a building block.
 - Some approaches to doing so are discussed in this section.
 - Although we refer to DES throughout the discussion, and DES is the most prominent instance where these techniques have been applied, everything we say here applies generically to any block cipher.
 - The first approach would be to somehow modify the internal structure of DES, while increasing the key length.
 - For example, one could leave the mangler function untouched and simply use a 128-bit master key with a different key schedule (still choosing a 48-bit sub-key in each round).
 - Or, one could change the S-boxes themselves and use a larger sub-key in each round.
 - The disadvantage of this approach is that by modifying DES — in even the smallest way — we lose the confidence we have gained in DES by virtue of the fact that it has remained secure for so many years.
 - More to the point, cryptographic constructions are very sensitive and thus even mild, seemingly insignificant changes can render the original construction completely insecure.
 - Changing the internals of a block cipher is therefore not recommended.

14 Lecture15

Double Encryption: Let F be a block cipher. Then a new block cipher F' with a key that is twice the length of the original one is defined by

$$F'_{k_1, k_2}(x) = F_{k_2}(F_{k_1}(x))$$

where k_1 and k_2 are independent keys. If F is DES then F' is a block cipher taking a 112-bit key. If exhaustive search were the best available attack on F' , a key length of 112 bits would be sufficient. Unfortunately, we now show an attack on F' that runs in time roughly 2^n when the original keys k_1 and k_2 are each of length n (and the block length is at least n); This is significantly less than the 2^{2n} time one would hope would be necessary to carry out an exhaustive search for a $2n$ -bit key. This means that the new block cipher is essentially no better than the old one, even though it has a key that is twice as long.

Meet-in-the-middle-attack: Say the adversary is given a single input/output pair (x, y) where $y = F'_{k_1, k_2}(x) = F_{k_2}(F_{k_1}(x))$. The adversary narrows down the set of possibilities this way:

ALGORITHM 7.ω

Meet-in-the-middle Attack

Input: (x, y)

Output: Set S of candidates key pairs (k_1, k_2)

Let $S := \emptyset$.

For each $k_1 \in \{0, 1\}^n$, **let** $z := F_{k_1}(x)$ and **store** (z, k_1) in list L .

For each $k_2 \in \{0, 1\}^n$, **let** $z := F_{k_2}^{-1}(y)$ and **store** (z, k_2) in list L' .

Sort L and L' , respectively, by their z components.

For each z such that $(z, k_1) \in L$ and $(z, k_2) \in L'$, add (k_1, k_2) to S .

Figure 52: A meet-in-the-middle attack

- The set S output by this algorithm contains exactly those values (k_1, k_2) for which $y = F'_{k_1, k_2}(x)$

- This holds because it outputs exactly those values (k_1, k_2) satisfying

$$F_{k_2}^{-1}(y) = F_{k_1}(x)$$

which holds if and only if $y = F'_{k_1, k_2}(x)$.

- If n is also the block length of F then a random pair (k_1, k_2) is expected to satisfy Equation $F_{k_2}^{-1}(y) = F_{k_1}(x)$ with probability roughly 2^{-n} and so the size of S will be approximately $2^{2n}/2^n = 2^n$.
- Given a few input/output pairs and intersecting the S 's obtained from these pairs is expected to identify the correct (k_1, k_2) with very high probability.

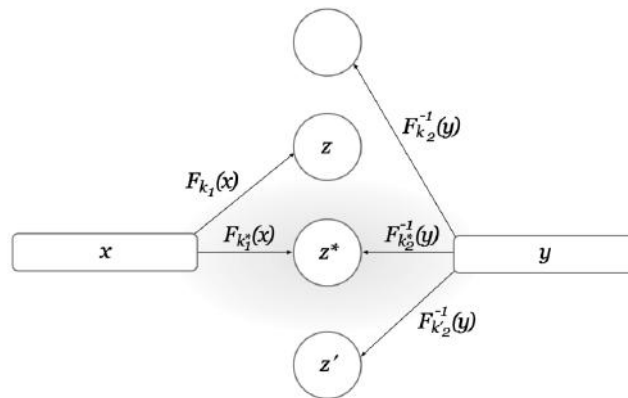


FIGURE 6.7: A meet-in-the-middle attack.

Figure 53: A meet-in-the-middle attack

Triple Encryption: The obvious generalization of the preceding approach is to apply the block cipher three times in succession. Two variants of this approach are common:

1. **Variant 1 - Three independent keys:** Choose 3 independent keys k_1, k_2, k_3 and define

$$F'_{k_1, k_2, k_3}(x) = F_{k_3}(F_{k_2}^{-1}(F_{k_1}(x)))$$

2. **Variant 2 - Two independent keys:** Choose 2 independent keys k_1, k_2 and define

$$F'_{k_1, k_2}(x) = F_{k_1}(F_{k_2}^{-1}(F_{k_1}(x)))$$

Before comparing the security of the two alternatives we note that the middle invocation of the original cipher is actually in the reverse direction. If F is a sufficiently good cipher this

makes no difference to the security, since if F is a strong pseudorandom permutation then F^{-1} must be too. The reason for this strange alternation between F , F^{-1} , and F is so that if one chooses $k_1 = k_2 = k_3$, the result is a single invocation of F with k_1 . This ensures backward compatibility (i.e., in order to switch back to a single invocation of F , it suffices to just set the keys to all be equal).

- **Security of the first variant:** Before comparing the security of the two alternatives we note that the middle invocation of the original cipher is actually in the reverse direction. If F is a sufficiently good cipher this makes no difference to the security, since if F is a strong pseudorandom permutation then F^{-1} must be too. If F is a sufficiently good cipher this makes no difference to the security, since if F is a strong pseudorandom permutation then F^{-1} must be too. The key length of this variant is $3n$ (where, as before, the key length of the original cipher F is n) and so we might hope that the best attack on this cipher would require time 2^{3n} . However, the cipher is susceptible to a meet-in-the-middle attack just as in the case of double encryption, though the attack now takes time 2^{2n} . This is the best known attack. Thus, although this variant is not as secure as we might have hoped, it obtains sufficient security for all practical purposes if $n = 56$.
- **Security of the second variant:** The key length of this variant is $2n$ and so the best we can hope for is security against attacks running in time 2^{2n} . There is no known attack with better time complexity when the adversary is given only a single input/output pair. However, there is a known chosen-plaintext attack that finds the key in time 2^n using 2^n chosen input/output pairs. Despite this, it is still a reasonable choice in practice.

Advanced Encryption Standard (AES)

- The AES block cipher has a 128-bit block length and can use 128-bit, 192-bit, or 256-bit keys.
- The length of the key affects the key schedule (i.e., the sub-key that is used in each round) as well as the number of rounds, but does not affect the high-level structure of each round.
- In contrast to DES that uses a Feistel structure, AES is essentially a substitution-permutation network.

- During computation of the AES algorithm, a 4-by-4 array of bytes called the state is modified in a series of rounds.
- The state is initially set equal to the input to the cipher (note that the input is 128 bits which is exactly 16 bytes)

AES Construction:

1. **Stage1 - AddRoundKey:** In every round of AES, a 128-bit sub-key is derived from the master key, and is interpreted as a 4-by-4 array of bytes. The state array is updated by XORing it with this subkey.

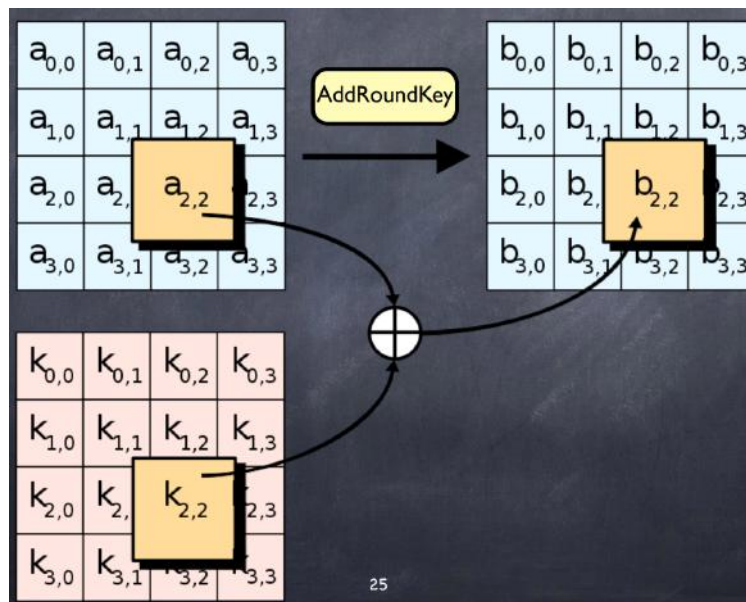


Figure 54: AddRoundKey

2. **Stage2 - SubBytes:** In this step, each byte of the state array is replaced by another byte according to a single fixed lookup table S . This S-box is a bijection over $\{0, 1\}^8$. We stress that there is only one S-box and it is used for substituting all the bytes in the state array in every round.

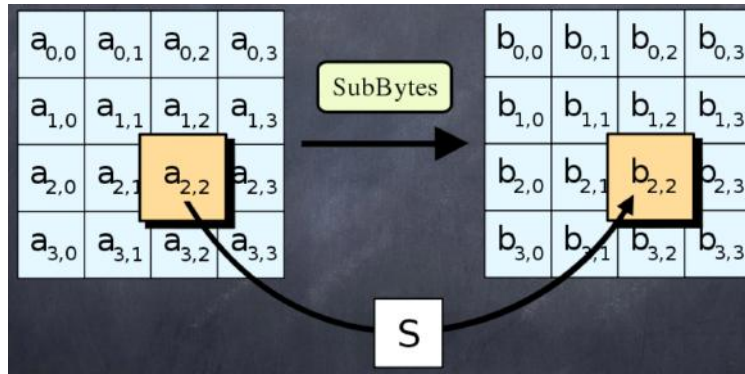


Figure 55: SubBytes

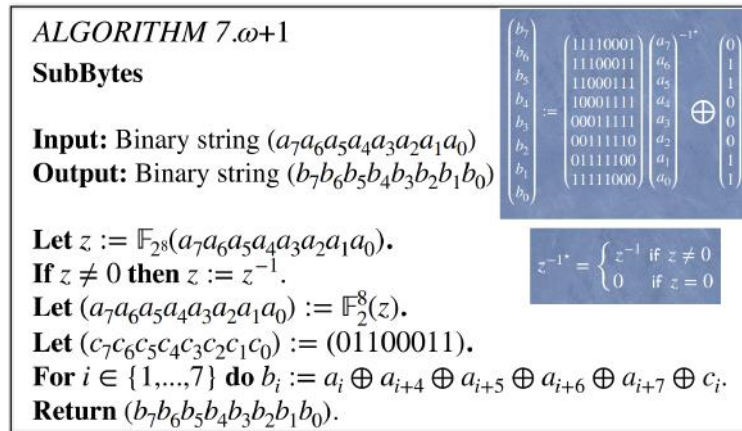


Figure 56: SubBytes

3. **Stage3 - ShiftRows:** In this step, the bytes in each row of the state array are cyclically shifted to the left as follows: the first row of the array is untouched, the second row is shifted one place to the left, the third row is shifted two places to the left, and the fourth row is shifted three places to the left. (All shifts are cyclic.)

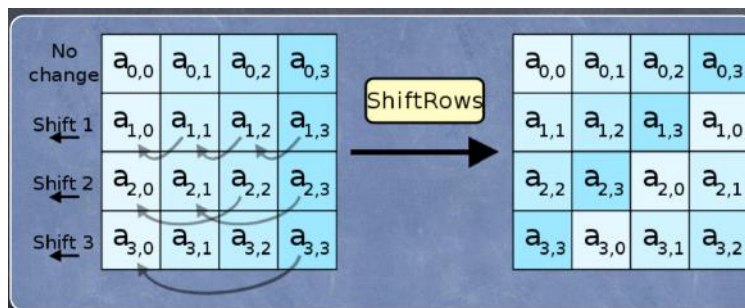


Figure 57: ShiftRows

4. **Stage4 - MixColumns:** In this step, an invertible linear transformation is applied to each column. One can think of this as matrix multiplication (over the appropriate field).

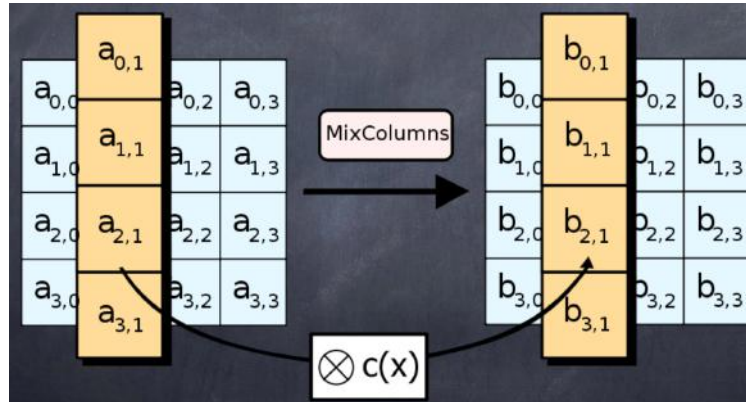


Figure 58: MixColumns

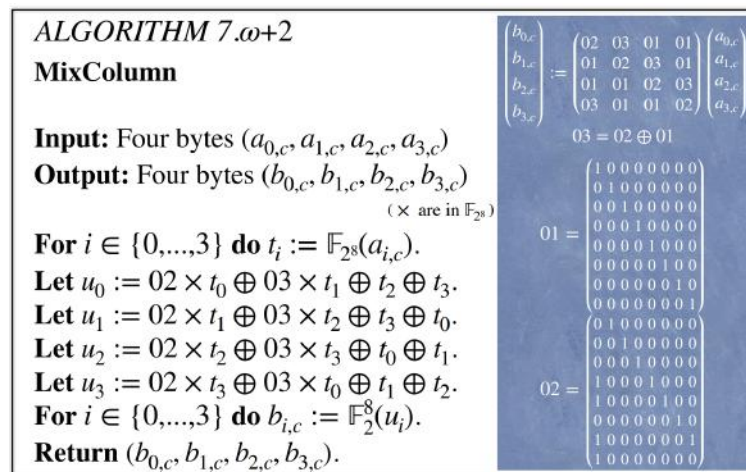


Figure 59: MixColumns

- By viewing stages 3 and 4 together as a “mixing” step, we see that each round of AES has the structure of a substitution-permutation network:
- The round sub-key is first XORed with the input to the current round;
- Next, a small, invertible function is applied to “chunks” of the resulting value;
- Finally, the bits of the result are mixed in order to obtain diffusion.

- The only difference is that, unlike our general description of substitution-permutation networks, here the mixing step does not consist of a simple permutation of the bits but is instead carried out using an invertible linear transformation of the bits.
- Simplifying things a bit and looking at a trivial 3-bit example, an invertible linear transformation might map x to $\langle x_1 \oplus x_2 || x_2 \oplus x_3 || x_1 \oplus x_2 \oplus x_3 \rangle$
- The number of rounds in AES depends on the length of the key.
- There are 10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key.
- In the final round of AES the MixColumns stage is replaced with an additional AddRoundKey step.
- We stress that the existing attacks are for reducedround variants of AES, and as of today no attack better than exhaustive key search is known for the full AES construction.

15 Lecture16

Key Distribution Center (KDC): An approach is to rely on the fact that all employees may trust some entity — say, the IT manager of the organization — at least with respect to the security of work-related information. It is therefore possible for the IT manager to set up a single server, called a key distribution center (KDC), that can act as an intermediary between employees that wish to communicate. A KDC can work in the following way:

- First, all employees share a single key with the KDC; this key can be generated and shared, e.g., on the employee's first day at work.
- Then, when employee Alice wants to communicate securely with employee Bob, she sends a message to the KDC saying 'Alice wishes to communicate with Bob' (where this message is authenticated using the key shared by Alice and the KDC).
- The KDC then chooses a new random secret key, called a session key, and sends this key to Alice encrypted using Alice's key, and also to Bob encrypted using Bob's key.
- Once Alice and Bob recover the session key, they can use it to communicate securely.

- When they are done with their conversation, they can (and should) erase this key because they can always contact the server again should they wish to communicate again at some later time.

Advantages of KDC:

1. Each employee needs to store only one secret key and so a smartcard-type solution can be deployed. It is true that the KDC needs to store many keys. However, the KDC can be secured in a safe place and given the highest possible protection against network attacks.
2. When an employee joins the organization all that must be done is to set up a secret key between this employee and the KDC. No other employees need to update the set of keys they hold. The same is true when an employee leaves the organization.

Disadvantages of KDC:

1. A successful attack on the KDC will result in a complete break of the system for all parties. Thus, the motivation to break into the KDC is very great, increasing the security risk. In addition, an adversary internal to the organization who has access to the KDC (for example, the IT manager) can decrypt all communication between all parties.
2. The KDC is a single point of failure: if the KDC crashes, secure communication is temporarily impossible. Since all employees are continually contacting the KDC, the load on the KDC can be very high thereby increasing the chances that it may fall or be slow to respond. A simple solution is to replicate the KDC. This works but the existence of more KDCs means that there are now more points of attack on the system. Furthermore, it becomes more difficult to add or remove employees, since updates must be securely propagated to all KDCs.

Protocols for key distribution using a KDC:

- There are a number of protocols that can be found in the literature for secure key distribution using a KDC. One of these is the classic Needham-Schroeder protocol. We will not go into the details of this protocol. We do mention one engineering feature of the protocol.

- When Alice contacts the KDC and asks to communicate with Bob, the KDC does not send the encrypted session key to both Alice and Bob.
- Rather, the KDC sends the session key encrypted under both Alice's and Bob's keys to Alice, and Alice herself forwards to Bob the session key encrypted under his key

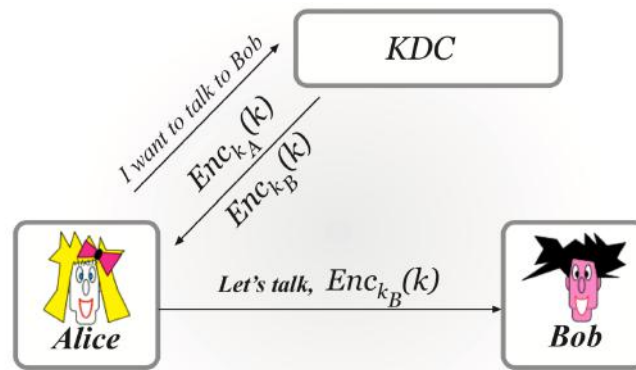


Figure 60: KDC

- The protocol was designed in this way due to the fact that Bob may not be online; this could potentially cause a problem for the KDC who might “hang” indefinitely waiting for Bob to respond.
- By sending both encrypted keys to Alice, the KDC is relieved of maintaining an open session.
- The session key encrypted under Bob's key that the KDC sends to Alice is called a ticket, and can be viewed as a credential allowing Alice to talk to Bob.

Diffie-Hellman Key Exchange

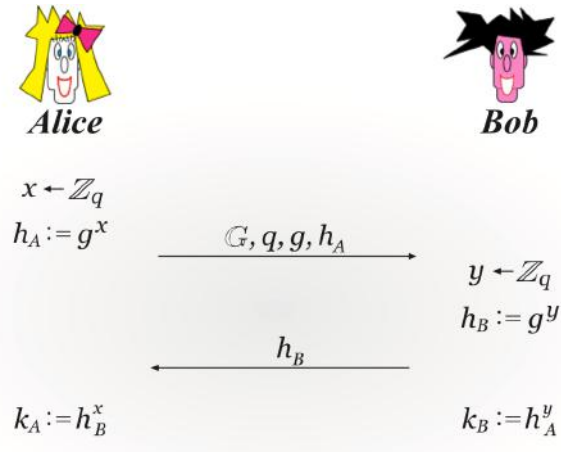


Figure 61: The Diffie-Hellman key-exchange protocol

Public-Key Primitives: An interactive key-exchange protocol is a method whereby parties who do not share any secret information can generate a shared, secret key by communicating over a public channel. The main property guaranteed here is that an eavesdropping adversary who sees all the messages sent over the communication line does not learn anything about the resulting secret key.

Public Key Exchange

The key-exchange experiment $\text{KE}_{\mathcal{A}, \Pi}^{\text{eav}}(n)$:

1. Two parties holding 1^n execute protocol Π . This results in a transcript trans containing all the messages sent by the parties, and a key k output by each of the parties.
2. A random bit $b \leftarrow \{0, 1\}$ is chosen. If $b = 0$ set $\hat{k} := k$, and if $b = 1$ then choose $\hat{k} \leftarrow \{0, 1\}^n$ uniformly at random.
3. \mathcal{A} is given trans and \hat{k} , and outputs a bit b' .
4. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. (In case $\text{KE}_{\mathcal{A}, \Pi}^{\text{eav}}(n) = 1$, we say that \mathcal{A} succeeds.)

Figure 62

Theorem: A key-exchange protocol Π is secure in the presence of an eavesdropper if for every PPT adversary \mathcal{A} there exists a negligible function negl such that

$$\Pr[\text{KE}_{\mathcal{A}, \Pi}^{\text{eav}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

Diffie-Hellman Key Exchange

CONSTRUCTION 11.2

- **Common input:** The security parameter 1^n
- **The protocol:**
 1. Alice runs $\mathcal{G}(1^n)$ to obtain (\mathbb{G}, q, g) .
 2. Alice chooses a uniform $x \in \mathbb{Z}_q$, and computes $h_A := g^x$.
 3. Alice sends (\mathbb{G}, q, g, h_A) to Bob.
 4. Bob receives (\mathbb{G}, q, g, h_A) . He chooses a uniform $y \in \mathbb{Z}_q$, and computes $h_B := g^y$. Bob sends h_B to Alice and outputs the key $k_B := h_A^y$.
 5. Alice receives h_B and outputs the key $k_A := h_B^x$.

Figure 63: Construction of Diffie-Hellman Key Exchange

Computational Diffie-Hellman Problem (CDH): Fix a cyclic group \mathcal{G} and a generator $g \in \mathcal{G}$. Given two group elements h_1, h_2 , define

$$DH_g(h_1, h_2) = g^{\log_g h_1 \cdot \log_g h_2}$$

That is, if $h_1 = g^x$ and $h_2 = g^y$ then

$$DH_g(g^x, g^y) = g^{x \cdot y} = h_1^y = h_2^x$$

The CDH problem is to compute $DH_g(h_1, h_2)$ given randomly-chosen h_1 and h_2 .

- If the discrete logarithm problem relative to some \mathcal{G} is easy, then the CDH problem is, too: given h_1 and h_2 , first compute $x = \log_g h_1$ and output the answer h_2^x
- In contrast, it is not clear whether hardness of the discrete logarithm problem necessarily implies that the CDH problem is hard as well.

Decisional Diffie-Hellman Problem (DDH): The DDH problem, roughly speaking, is to distinguish $DH_g(h_1, h_2)$ from a random group element for randomly-chosen (h_1, h_2) . That is, given randomly-chosen (h_1, h_2) and a candidate solution H' , the problem is to decide whether

$$H' = DH_g(h_1, h_2)$$

or whether H was chosen randomly from \mathcal{G} .

The Discrete Logarithm and Diffie-Hellman Assumptions Definitions: We say that the DDH problem is hard relative to \mathcal{G} if for all probabilistic polynomial-time algorithms A there exists a negligible function negl such that

$$|Pr[A(\mathcal{G}, q, g, g^x, g^y, g^z) = 1] - Pr[A(\mathcal{G}, q, g, g^x, g^y, g^{xy}) = 1]| \leq \text{negl}(n)$$

where in each case the probabilities are taken over the experiment in which $\text{Gen}(1^n)$ outputs (\mathcal{G}, q, g) , and then random $x, y, z \in_R Z_q$ are chosen.

Theorem: If the decisional Diffie-Hellman problem is hard relative to \mathcal{G} , then the Diffie-Hellman key-exchange protocol Π is secure in the presence of an eavesdropper.

Active Adversaries: Although eavesdropping attacks are by far the most common (as they are so easy to carry out), they are by no means the only possible attack. Active attacks, in which the adversary sends messages of its own to one or both of the parties are also a concern, and any protocol used in practice must be resilient to active attacks.

- **impersonation:** attacks where only one of the honest parties is executing the protocol and the adversary impersonates the other party
- **man-in-the-middle:** attacks where both honest parties are executing the protocol and the adversary is intercepting and modifying messages being sent from one party to the other.

It is worth remarking that the Diffie-Hellman protocol is completely insecure against man-in-the-middle attacks. In fact, a man-in-the-middle adversary can act in such a way that Alice and Bob terminate the protocol with different keys k_A, k_B that are both known to the adversary, yet neither Alice nor Bob can detect that any attack was carried out.

The Public-Key Revolution: In a public-key encryption scheme the encryption key is called the public key, since it is publicized by the receiver so that anyone who wishes to send an encrypted message may do so, and the decryption key is called the private key since it is kept completely private by the receiver.

1. Public-key encryption allows key distribution to be done over public channels. This can potentially simplify initial deployment of the system, and can also ease maintenance of the system when parties join or leave.

2. Public-key encryption vastly reduces the need to store many secret keys. Even if all pairs of parties want the ability to communicate securely, each party need only store his own private key in a secure fashion. Other parties' public keys can either be obtained when needed, or stored in a non-secure (i.e., publicly-readable) fashion.
3. Finally, public-key cryptography is (more) suitable for open environments where parties who have never previously interacted want the ability to communicate securely. For example, a merchant can post their public key on-line; any user making a purchase can obtain the merchant's public key, as needed, when they need to encrypt their credit card information.

16 Lecture17

Key Encapsulation Mechanism (KEM):

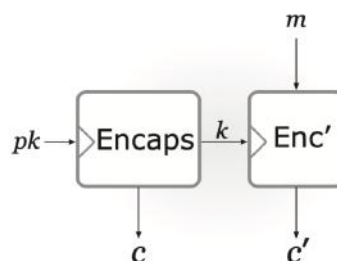
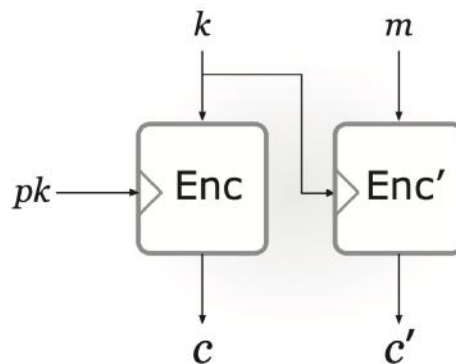


FIGURE 11.2: Hybrid encryption using a KEM.

Figure 65: Key Encapsulation Mechanism (KEM)

DEFINITION 12.9 A key-encapsulation mechanism (KEM) is a tuple of probabilistic polynomial-time algorithms $(\text{Gen}, \text{Encaps}, \text{Decaps})$ such that:

1. The key-generation algorithm Gen takes as input the security parameter 1^n and outputs a public-/private-key pair (pk, sk) . We assume pk and sk each have length at least n , and that n can be determined from pk .
2. The encapsulation algorithm Encaps takes as input a public key pk and the security parameter 1^n . It outputs a ciphertext c and a key $k \in \{0, 1\}^{\ell(n)}$ where ℓ is the key length. We write this as $(c, k) \leftarrow \text{Encaps}_{pk}(1^n)$.
3. The deterministic decapsulation algorithm Decaps takes as input a private key sk and a ciphertext c , and outputs a key k or a special symbol \perp denoting failure. We write this as $k := \text{Decaps}_{sk}(c)$.

Figure 66: Key Encapsulation Mechanism (KEM)

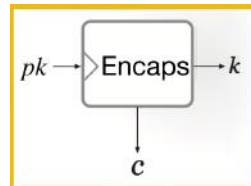


Figure 67: Key Encapsulation Mechanism (KEM)

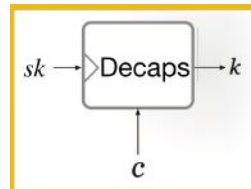


Figure 68: Key Encapsulation Mechanism (KEM)

CPA-security of Key-encapsulation mechanism

The eavesdropping indistinguishability experiment $\text{KEM}_{\mathcal{A},\Pi}^{\text{eav}}(n)$:

1. $\text{Gen}(1^n)$ is run to obtain keys (pk, sk) . Then $\text{Encaps}_{pk}(1^n)$ is run to generate (c, k) with $k \in \{0, 1\}^n$.
2. A random bit $b \leftarrow \{0, 1\}$ is chosen. If $b = 0$ set $\hat{k} := k$. If $b = 1$ then choose $\hat{k} \leftarrow \{0, 1\}^n$.
3. Give (pk, c, \hat{k}) to \mathcal{A} , who outputs a bit b' . The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Figure 69: The eavesdropping indistinguishability experiment $\text{KEM}_{\mathcal{A},\Pi}^{\text{eav}}(n)$

DEFINITION 11.11 Key-encapsulation mechanism Π is CPA-secure if for all probabilistic polynomial-time adversaries \mathcal{A} there is a negligible function negl such that

$$\Pr[\text{KEM}_{\mathcal{A},\Pi}^{\text{eav}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

Figure 70: CPA-Secure of KEM

Hybrid Encryption:

CONSTRUCTION 11.10

Let $\Pi = (\text{Gen}, \text{Encaps}, \text{Decaps})$ be a KEM with key length n , and let $\Pi' = (\text{Gen}', \text{Enc}', \text{Dec}')$ be a private-key encryption scheme. Construct a public-key encryption scheme $\Pi^{\text{hy}} = (\text{Gen}^{\text{hy}}, \text{Enc}^{\text{hy}}, \text{Dec}^{\text{hy}})$ as follows:

- Gen^{hy} : on input 1^n run $\text{Gen}(1^n)$ and use the public and private keys (pk, sk) that are output.
- Enc^{hy} : on input a public key pk and a message $m \in \{0, 1\}^*$ do:
 1. Compute $(c, k) \leftarrow \text{Encaps}_{pk}(1^n)$.
 2. Compute $c' \leftarrow \text{Enc}'_k(m)$.
 3. Output the ciphertext $\langle c, c' \rangle$.
- Dec^{hy} : on input a private key sk and a ciphertext $\langle c, c' \rangle$ do:
 1. Compute $k := \text{Decaps}_{sk}(c)$.
 2. Output the message $m := \text{Dec}'_k(c')$.

Figure 71: Construction of Hybrid Encryption

Theorem: If Π is a CPA-Secure KEM and Π' is a private-key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper, then Π^{hy} as in Construction

12.10 is a public-key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper (CPA-secure).

Chosen-Ciphertext Attacks for KEM $KEM_{A,\Pi}^{cca}(n)$:

The CCA indistinguishability experiment $KEM_{A,\Pi}^{cca}(n)$:

1. $\text{Gen}(1^n)$ is run to obtain keys (pk, sk) . Then $\text{Encaps}_{pk}(1^n)$ is run to generate (c, k) with $k \in \{0, 1\}^n$.
2. A random bit $b \leftarrow \{0, 1\}$ is chosen. If $b = 0$ set $\hat{k} := k$. If $b = 1$ then choose $\hat{k} \leftarrow \{0, 1\}^n$.
3. \mathcal{A} is given (pk, c, \hat{k}) and access to an oracle $\text{Decaps}_{sk}(\cdot)$, but may not request decapsulation of c itself.
4. \mathcal{A} outputs a bit b' . The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

DEFINITION 12.13 Key-encapsulation mechanism Π is CCA-secure if for all probabilistic polynomial-time adversaries \mathcal{A} there is a negligible function negl such that

$$\Pr[\text{KEM}_{A,\Pi}^{cca}(n) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

Figure 72: CCA-secure

Theorem: If Π is a CCA-secure KEM and Π' is a CCA-secure private-key encryption scheme, then Π^{hy} as in Construction 12.10 is a CCA-secure public-key encryption scheme.

RSA Encryption:

CONSTRUCTION 12.26

Let GenRSA be as in the text. Define a public-key encryption scheme as follows:

- **Gen:** on input 1^n run $\text{GenRSA}(1^n)$ to obtain N, e , and d . The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- **Enc:** on input a public key $pk = \langle N, e \rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the ciphertext

$$c := [m^e \bmod N].$$

- **Dec:** on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute the message

$$m := [c^d \bmod N].$$

The plain RSA encryption scheme.

Figure 73: RSA

ALGORITHM 12.25**RSA key generation GenRSA****Input:** Security parameter 1^n **Output:** N, e, d as described in the text $(N, p, q) \leftarrow \text{GenModulus}(1^n)$ $\phi(N) := (p-1)(q-1)$ **choose** $e > 1$ such that $\gcd(e, \phi(N)) = 1$ **compute** $d := [e^{-1} \bmod \phi(N)]$ **return** N, e, d

Figure 74: RSA Key Generation. (In Cocks' Variation, $e=N$ and therefore $d = N^{-1} \bmod \phi(N)$)

The RSA Assumption: The RSA problem can be describe informally as follows: given a modulus N , and integer (exponent) $e > 0$ that is relatively prime to $\phi(N)$, and an element $y \in Z_N^*$, compute $\sqrt[e]{y} \bmod N$; Given N, e, y find x such that $y = x^e \bmod N$.

The RSA experiment $RSA - inv_{A, GenRSA}(n)$:

1. Run $GenRSA(1^n)$ to obtain (N, e, d) .
2. Choose $y \leftarrow Z_N^*$
3. A is given N, e, y , and outputs $x \in Z_N^*$
4. The output of the experiment is defined to be 1 if $y = x^e \bmod N$, and 0 otherwise.

Definition: We say that the RSA problem is hard relative to GenRSA if for all probabilistic polynomial-time algorithms A there exists a negligible function negl such that

$$\Pr[RSA - inv_{A, GenRSA}(n) = 1] \leq \text{negl}(n)$$

RSA vs Factoring: The RSA assumption implies that $\phi(N)$ is unknown.

Theorem: Knowledge of N and $\phi(N)$ factors N .

Proof: $N = pq$, and $\phi(N) = (p-1)(q-1) = N - p - q + 1$

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

$$p + N/p = N - \phi(N) + 1 \text{ or } p^2 - (N - \phi(N) + 1)p + N = 0$$

p and q are the two solutions of the quadratic equation $ap^2 + bp + c = 0$ with $a = 1, b = \phi(N) - N + 1, c = N$

RSA vs Factoring: The RSA assumption implies that d is unknown.

Theorem: Knowledge of N, e, d factors N ,

Proof: Use the $RSA - FACOTR(N, ed - 1)$ algorithm, we can succeed with probability $\geq \frac{1}{2}$.

ALGORITHM 12.ω

RSA – FACTOR

Input: Integer modulus N , exponents e, d s.t. $ed \equiv 1 \pmod{\phi(N)}$

Output: Primes p, q s.t. $N = pq$

Let $2^s r = ed - 1$ with r odd.

Let $w \leftarrow \{1, \dots, N-1\}, x := \gcd(w, N)$.

If $1 < x < N$ **then Return** $(x, N/x)$.

Let $v := w^r \pmod{N}$.

If $v = 1$ **then Return** "Failure".

While $v \neq 1$ **do** $v' := v; v := v^2 \pmod{N}$.

If $v' = N - 1$ **then Return** "Failure".

Return $(\gcd(v' - 1, N), \gcd(v' + 1, N))$.

Figure 75: $RSA - FACOTR(N, ed - 1)$ algorithm

RSA Implementation Issues

- Encoding binary string w as elements $x \in Z_N^*$. Let $l = ||N||$. Any binary string w of length $l-1$ can be viewed as an element of Z_N in the natural way. It is also possible to encode strings of varying lengths (up to length $l-2$) as elements of Z_N by padding using some unambiguous padding scheme. For instance using $w \rightarrow NATURAL(1||w)$. Finally, the probability that the resulting $x \in Z_N$ is fortunately negligible if the binary string was selected independently from N .

- There does not appear to be any difference in the hardness of the RSA problem for different exponents e and, as such, different methods have been suggested for selecting e . One popular choice is to set $e = 3$, since then computing $e - th$ powers modulo N (as done when encrypting in the Plain RSA scheme) requires only two multiplications. If e is to be set equal to 3, then the primes p and q must be chosen to satisfy $p, q \not\equiv 1 \pmod{3}$ so that we have $\gcd(e, \phi(N)) = 1$.
- Note that choosing d to be small in order to speed up decryption (that is, changing GenRSA so that a small d is chosen first and then computing e) is a bad idea. If d is chosen in a small range (say, $d < 2^{16}$) then a brute-force search for d is easy to carry out. Even if d is chosen so that $d \leq \sqrt[4]{N}$, ($\|d\| \leq \|N\|/4$ and so brute-force attacks are ruled out) there is another attack that can be used to recover d from the public key (N, e) .

Equivalent Key Length	RSA	Discrete Logarithm	
	Modulus Length	Order- q Subgroup of \mathbb{Z}_p^*	Elliptic-Curve Group Size q
112	2048	p : 2048, q : 224	224
128	3072	p : 3072, q : 256	256
192	7680	p : 7680, q : 384	384
256	15360	p : 15360, q : 512	512

Figure 76

Attacks on Plain RSA:

- Encrypting short messages using small e . If e is small then the encryption of “small” messages is insecure when using plain RSA encryption.
- For example, say $e = 3$ and the message m is such that $m < \sqrt[3]{N}$ (or $\|m\| < \|N\|/3$) but m is otherwise unknown to an attacker. In this case, encryption of m does not involve any modular reduction since the integer m^3 is less than N .
- This means that given the ciphertext $c = m^3 \bmod N$ an attacker can determine m by computing $m = \sqrt[3]{c}$ over the integers, a computation that can be easily carried out.
- The above attack shows that short messages can be recovered easily from their encryption if plain RSA with small e is used. Here, we extend the attack to the case of arbitrary length messages as long as the same message is sent to multiple receivers.

- Let $e = 3$ as before, and say the same message m is sent to three different parties holding public keys $pk_1 = (N_1, 3)$, $pk_2 = (N_2, 3)$, and $pk_3 = (N_3, 3)$, respectively. Then an eavesdropper sees three ciphertexts $c_1 = m^3 \bmod N_1$, $c_2 = m^3 \bmod N_2$, $c_3 = m^3 \bmod N_3$.
- Let $N^* = N_1 \cdot N_2 \cdot N_3$. An extended version of Chinese remainder theorem says that there exists a unique non-negative value $C^* < N^*$ such that:

$$- C^* \equiv c_1 \bmod N_1$$

$$- C^* \equiv c_2 \bmod N_2$$

$$- C^* \equiv c_3 \bmod N_3$$

RSA Chosen-Ciphertext Attacks: Plain RSA encryption, say an adversary A intercepts the ciphertext $c = m^e \bmod N$.

- Then the adversary can choose a random $r \leftarrow \mathbb{Z}_N^*$ and compute the ciphertext $c' = r^e \cdot c \bmod N$.
- Given the (oracle) decryption m' of c' , A can easily recover $m = r^{-1} \cdot m' \bmod N$.

17 Lecture18

Padded RSA:

CONSTRUCTION 12.29

Let GenRSA be as before, and let ℓ be a function with $\ell(n) \leq 2n - 4$ for all n . Define a public-key encryption scheme as follows:

- **Gen:** on input 1^n , run $\text{GenRSA}(1^n)$ to obtain (N, e, d) . Output the public key $pk = \langle N, e \rangle$, and the private key $sk = \langle N, d \rangle$.
- **Enc:** on input a public key $pk = \langle N, e \rangle$ and a message $m \in \{0, 1\}^{\|N\| - \ell(n) - 2}$, choose a random string $r \leftarrow \{0, 1\}^{\ell(n)}$ and interpret $\hat{m} := 1 \| r \| m$ as an element of \mathbb{Z}_N^* . Output the ciphertext

$$c := [\hat{m}^e \bmod N].$$

- **Dec:** on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute

$$\hat{m} := [c^d \bmod N],$$

and output the $\|N\| - \ell(n) - 2$ least-significant bits of \hat{m} .

Figure 77

Theorem: If the RSA problem is hard relative to GenRSA then construction 12.29 with $l(n)$ is $O(\log n)$ has indistinguishable encryptions under a chosen-plaintext attack.

CONSTRUCTION 12.33

Let GenRSA be as usual, and define a KEM as follows:

- **Gen:** on input 1^n , run GenRSA(1^n) to obtain (N, e, d) . Then compute $d' := [d^n \bmod \phi(N)]$ (note that $\phi(N)$ could be computed from (N, e, d) , or obtained during the course of running GenRSA). Output $pk = \langle N, e \rangle$ and $sk = \langle N, d' \rangle$.
- **Encaps:** on input $pk = \langle N, e \rangle$ and 1^n , choose uniform $c_1 \in \mathbb{Z}_N^*$. Then for $i = 1, \dots, n$ do:

1. Compute $k_i := \text{lsb}(c_i)$.
2. Compute $c_{i+1} := [c_i^e \bmod N]$.

Output the ciphertext c_{n+1} and the key $k = k_1 \dots k_n$.

- **Decaps:** on input $sk = \langle N, d' \rangle$ and a ciphertext c , compute $c_1 := [c^{d'} \bmod N]$. Then for $i = 1, \dots, n$ do:
1. Compute $k_i := \text{lsb}(c_i)$.
 2. Compute $c_{i+1} := [c_i^e \bmod N]$.

Output the key $k = k_1 \dots k_n$.

Figure 78: $e = 3$ makes it very efficient

Theorem: If the RSA problem is hard relative to GenRSA then Construction 12.33 is a CPA-secure KEM.

PKCS

- A widely-used and standardized encryption scheme, RSA Laboratories Public-Key Cryptography Standard (PKCS) #1 version 1.5, utilizes what is essentially padded RSA encryption.
- For a public key $pk = (N, e)$ of the usual form, let k denote the length of N in bytes; i.e., k is the integer satisfying $2^{8k-1} \leq N < 2^{8k}$.
- Messages m to be encrypted are assumed to be a multiple of 8 bits long, and can have length up to $k-11$ bytes. Encryption of a message m that is D -bytes long is computed as $(00000000 || 00000010 || r || 00000000 || m)^e \bmod N$, where r is a randomly-generated string of $(k - D - 3)$ bytes, with none of these bytes equal to 0.
- PKCS #1 v1.5 is believed to be CPA-secure, although no proof solely based on the RSA assumption has ever been shown.

- Subsequent to the introduction of PKCS #1 v1.5, a chosen-ciphertext attack on this scheme was demonstrated.
- This motivated a change in the standard to a newer scheme called OAEP (for Optimal Asymmetric Encryption Padding).

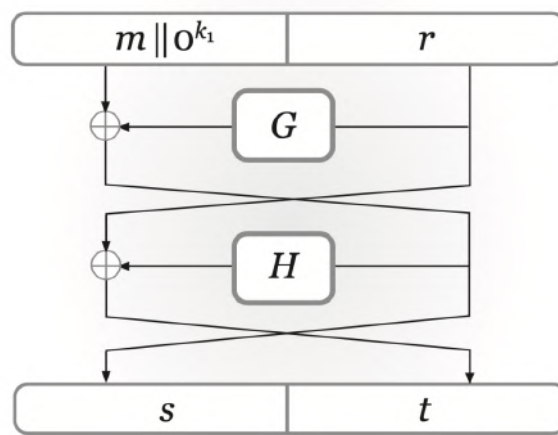


Figure 79: The OAEP padding mechanism

CONSTRUCTION 12.38

Let GenRSA be as in the previous sections, and ℓ, k_0, k_1 be as described in the text. Let $G : \{0, 1\}^{k_0} \rightarrow \{0, 1\}^{\ell+k_1}$ and $H : \{0, 1\}^{\ell+k_1} \rightarrow \{0, 1\}^{k_0}$ be functions. Construct a public-key encryption scheme as follows:

- **Gen:** on input 1^n , run $\text{GenRSA}(1^n)$ to obtain (N, e, d) . The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- **Enc:** on input a public key $\langle N, e \rangle$ and a message $m \in \{0, 1\}^\ell$, set $m' := m \parallel 0^{k_1}$ and choose uniform $r \in \{0, 1\}^{k_0}$. Then compute

$$s := m' \oplus G(r), \quad t := r \oplus H(s)$$

and set $\hat{m} := s \parallel t$. Output the ciphertext $c := [\hat{m}^e \bmod N]$.

- **Dec:** on input a private key $\langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute $\hat{m} := [c^d \bmod N]$. If $\|\hat{m}\| > \ell + k_0 + k_1$, output \perp . Otherwise, parse \hat{m} as $s \parallel t$ with $s \in \{0, 1\}^{\ell+k_1}$ and $t \in \{0, 1\}^{k_0}$. Compute $r := H(s) \oplus t$ and $m' := G(r) \oplus s$. If the least-significant k_1 bits of m' are not all 0, output \perp . Otherwise, output the ℓ most-significant bits of \hat{m} .

Figure 80: PKCS #1 v2.0

Elgamal Encryption Scheme

CONSTRUCTION 12.16

Let \mathcal{G} be as in the text. Define a public-key encryption scheme as follows:

- **Gen:** on input 1^n run $\mathcal{G}(1^n)$ to obtain (\mathbb{G}, q, g) . Then choose a uniform $x \leftarrow \mathbb{Z}_q$ and compute $h := g^x$. The public key is $\langle \mathbb{G}, q, g, h \rangle$ and the private key is $\langle \mathbb{G}, q, g, x \rangle$. The message space is \mathbb{G} .
- **Enc:** on input a public key $pk = \langle \mathbb{G}, q, g, h \rangle$ and a message $m \in \mathbb{G}$, choose a uniform $y \leftarrow \mathbb{Z}_q$ and output the ciphertext

$$\langle g^y, h^y \cdot m \rangle.$$
- **Dec:** on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $\langle c_1, c_2 \rangle$, output

$$\hat{m} := c_2 / c_1^x.$$

Figure 81: Elgamal Encryption Scheme

Theorem: If the DDH problem is hard relative to \mathcal{G} , then the Elgamal encryption scheme has indistinguishable encryptions under a chosen-plaintext attack.

Elgamal Implementation Issues

- Encoding binary strings. Let q be a Sophie Germain prime, i.e., q and $p = 2q + 1$ are both prime.
- Then the set of quadratic residues modulo p forms a group \mathcal{G} of order $q = (p - 1)/2$ under \times modulo p .
- We can map the integers $\bar{m} \in \{1, \dots, q\}$ to the set of quadratic residues modulo p by squaring: that is, the integer \bar{m} is mapped to the quadratic residue

$$m = \bar{m}^2 \bmod p$$

- This encoding is one-to-one and efficiently computable and reversible. (When extracting square roots modulo p , take the smaller square root i.e. $\min\{r, p - r\} \leq q$.)
- Given the above, we can map a new string w of length $\|q\| - 1$ to an element $m \in \mathcal{G}$ in the following way:

- given a string $w \in \{0, 1\}^{\|q\|-1}$
- interpret it as an integer in the natural way and add 1 to obtain an integer \bar{m} with $1 \leq \bar{m} \leq q$
- Then take $m = \bar{m}^2 \bmod p$

CONSTRUCTION 12.19

Let \mathcal{G} be as in the text. Define a KEM as follows:

- **Gen:** on input 1^n run $\mathcal{G}(1^n)$ to obtain (\mathbb{G}, q, g) . Choose uniform $x \leftarrow \mathbb{Z}_q$ and set $h := g^x$. Also specify a function $H : \mathbb{G} \rightarrow \{0, 1\}^{\ell(n)}$ for some function ℓ (see text). The public key is $\langle \mathbb{G}, q, g, h, H \rangle$ and the private key is $\langle \mathbb{G}, q, g, x \rangle$.
- **Encaps:** on input a public key $pk = \langle \mathbb{G}, q, g, h, H \rangle$ choose uniform $y \leftarrow \mathbb{Z}_q$ and output the ciphertext g^y and the key $H(h^y)$.
- **Decaps:** on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $c \in \mathbb{G}$, output the key $H(c^x)$.

Figure 82: An "El Gamal-like" KEM

THEOREM 12.20 *If the CDH problem is hard relative to \mathcal{G} , and H is modeled as a random oracle, then Construction 12.19 is CPA-secure.*

Figure 83

Elgamal Chosen-Ciphertext Attacks: Elgamal encryption. Say an adversary A intercepts a ciphertext $c = (c_1, c_2)$ that is an encryption of the (encoded) message m with respect to the public key $pk = (\mathcal{G}, q, g, h)$.

- This means that $c_1 = g^y$ and $c_2 = h^y \cdot m$ for some $y \in \mathbb{Z}_q$ unknown to A . Nevertheless, if the adversary computes $c'_2 = c_2 \cdot m'$ then it is easy to see that the ciphertext $c' = (c_1, c'_2)$ is an encryption of the message $m \cdot m'$.
- One might object that the receiver will become suspicious if it receives two ciphertexts c, c' that share the same first component. However, this is easy for the adversary to avoid.
- Letting c_1, c_2, m be as above, A can choose a random $y'' \leftarrow \mathbb{Z}_q$ and set $c''_1 = g^{y''} \cdot c_1$ and $c''_2 = h^{y''} \cdot c_2$.
- Then $c''_1 = g^{y''} \cdot g^y = g^{y+y''}$ and $c''_2 = h^{y''} \cdot h^y m = h^{y+y''} m$, and so the ciphertext $c'' = (c''_1, c''_2)$ is an encryption of m but completely random.

Post-Quantum Cryptography

18 Lecture19

Post-Quantum Cryptography

19 Lecture20

Definiton of Message Authentication Code (MAC): A message authentication code is a tuple of PPT algorithms $(\text{Gen}, \text{Mac}, \text{Vrfy})$ such that:

1. The key-generation algorithm Gen takes as input the security parameter 1^n and outputs a key k with $|k| \leq n$.
2. The tag-generation algorithm Mac takes as input a key k and a message $m \in \{0, 1\}^*$, and outputs a tag t . Since this algorithm may be randomized, we write this as $t \leftarrow \text{Mac}_k(m)$.
3. The verification algorithm Vrfy takes as input a key k , a message m , and a tag t . It outputs a bit b , with $b = 1$ meaning valid and $b = 0$ meaning invalid. We assume without loss of generality that Vrfy is deterministic, and so write this as $b = \text{Vrfy}_k(m, t)$.

It is required that for every n , every key k output by $\text{Gen}(1^n)$, and every $m \in \{0, 1\}^*$, it holds that

$$\text{Vrfy}_k(m, \text{Mac}_k(m)) = 1$$

If $(\text{Gen}, \text{Mac}, \text{Vrfy})$ is such that for every k output by $\text{Gen}(1^n)$, algorithm Mac_k is only defined for messages $m \in \{0, 1\}^{l(n)}$ (and Vrfy_k outputs 0 for any $m \notin \{0, 1\}^{l(n)}$), then we say that it is a fixed-length MAC for length $l(n)$.

Security of MAC $\text{Mac} - \text{Forge}_{A, \Pi}(n)$:

1. A random key k is generated by running $\text{Gen}(1^n)$
2. The adversary A is given 1^n and oracle access to $\text{Mac}_k(\cdot)$. The adversary eventually outputs a pair (m, t) . Let \mathcal{Q} denote the set of all queries that A asked to its oracle.
3. The output of the experiment is defined to be 1 if and only if

$$(a) \text{ Vrfy}_k(m, t) = 1$$

$$(b) m \notin \mathcal{Q}$$

Security of MAC: A message authentication code $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all PPT adver-

series A . there exists a negligible function negl such that

$$\Pr[\text{Mac} - \text{Forge}_{A,\Pi}(n) = 1] \leq \text{negl}(n)$$

The above definition is rather strong, in two respects.

- First, the adversary is allowed to request a MAC tag for any message of its choice.
- Second, the adversary is considered to have “broken” the scheme if it can output a valid tag on any previously-unauthenticated message.

Replay Attacks

- We emphasize that the previous definition, and message authentication codes in general, offer no protection against replay attacks in which a previously-sent message (and its MAC tag) are replayed to one of the honest parties. Nevertheless, replay attacks are a serious concern.
- Despite the real threat due to replay attacks, a MAC inherently cannot protect against such attacks since the definition of a MAC (Definition 4.1) does not incorporate any notion of state into the verification algorithm (and so every time a valid pair (m, t) is presented to the verification algorithm, it will always output 1). Rather, protection against replay attacks — if such protection is necessary at all — is left to some higher level application.

Two common techniques for preventing replay attacks involve the use of sequence numbers or time-stamps. The basic idea of the first approach is that each message m is assigned a sequence number i , and the MAC tag is computed over the concatenated message $i||m$. It is assumed here that the sender always assigns a unique sequence number to each message, and that the receiver keeps track of which sequence numbers it has already seen.

- **sequence numbers:** Now, any successful replay of a message m will have to forge a valid MAC tag on a new concatenated message $i'||m$, where i' has never been used before. This is ruled out by the security of the MAC. A disadvantage of using sequence numbers is that the receiver must store a list of all previous sequence numbers it has received.

- **time-stamps:** Here, the sender essentially appends the current time to the message (say, to the nearest ms) rather than a sequence number. When the receiver obtains a message, it checks whether the included time-stamp is within some acceptable window of the current time. This method has certain drawbacks as well, including the need for the sender and receiver to maintain closely synchronized clocks, and the possibility that a replay attack can still take place as long as it is done quickly enough.

One-time message authentication $Mac - Forge_{A,\Pi}^{1-time}(n)$:

The one-time message authentication experiment $Mac-forge_{A,\Pi}^{1-time}$:

1. A random key k is generated by running Gen .
2. The adversary A outputs a message m' , and is given in return a tag $t' \leftarrow Mac_k(m')$.
3. A outputs (m, t) .
4. The output of the experiment is defined to be 1 if and only if **(1)** $Vrfy_k(m, t) = 1$ and **(2)** $m \neq m'$.

Figure 84: One-time message authentication

Perfect Security of MAC: A message authentication code $\Pi = (Gen, Mac, Vrfy)$ is ϵ -existentially unforgeable under a single chosen-message attack, or just ϵ -1-time-secure, if for all adversaries A :

$$Pr[Mac - Forge_{A,\Pi}^{1-time}(n) = 1] \leq \epsilon$$

The syntax of an it-MAC: A message authentication code (or MAC) is a tuple of PPT algorithms $(Gen, Mac, Vrfy)$ such that:

1. The key-generation algorithm Gen outputs a random key $k \in \mathcal{K}$.
2. The tag-generation algorithm Mac takes as input a key k and a message $m \in \{0, 1\}^*$, and outputs a tag t . Since this algorithm may be randomized, we write this as $t \leftarrow Mac_k(m)$.
3. The verification algorithm $Vrfy$ takes as input a key k , a message m , and a tag t . It outputs a bit b , with $b = 1$ meaning valid and $b = 0$ meaning invalid. We assume without loss of generality that $Vrfy$ is deterministic, and so write this as $b = Vrfy_k(m, t)$.

$$Mac - Forge_{A,\Pi}^{it}(n)$$

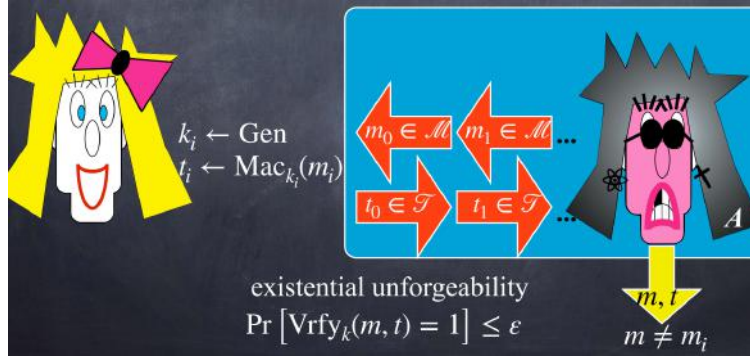


Figure 85: $Mac - Forge_{A,\Pi}^{it}(n)$

Security of it-MAC: A message authentication code $\Pi = (Gen, Mac, Vrfy)$ is ϵ -existentially unforgeable under an adaptive chosen-message attack, or just ϵ -secure, if for all adversaries A :

$$\Pr[Mac - Forge_{A,\Pi}^{it}(n) = 1] \leq \epsilon$$

Existential Unforgeability: Let $\Pi = (Gen, Mac, Vrfy)$ be an authentication scheme over a message space \mathcal{M} . Π is perfectly secure with respect to Definition of Security of it-MAC if it is perfectly secure with respect to the definition of Perfect Security of MAC and a new key is generated for each authentication.

One-Time MACs from Strongly Universal Functions: A function $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ is strongly universal if for all distinct $m, m' \in \mathcal{M}$ and all (not necessarily distinct) $t, t' \in \mathcal{T}$ it holds that

$$\Pr[h_k(m) = t \wedge h_k(m') = t'] = \frac{1}{|\mathcal{T}|^2}$$

where the probability is taken over uniform choice of $k \in \mathcal{K}$.

CONSTRUCTION 4.20

Let $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ be a strongly universal function. Define a MAC for messages in \mathcal{M} as follows:

- **Gen**: choose uniform $k \in \mathcal{K}$ and output it as the key.
- **Mac**: on input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, output the tag $t := h_k(m)$.
- **Vrfy**: on input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$, output 1 if and only if $t \stackrel{?}{=} h_k(m)$. (If $m \notin \mathcal{M}$, then output 0.)

Figure 86

Theorem: Let $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ be a strongly universal function. Then Construction 4.20 is a $1/|\mathcal{T}|$ -secure one-time MAC for messages in \mathcal{M} .

We first define the following function:

$$h : Z_p^2 \times Z_p \rightarrow Z_p, h_{a,b}(m) = am + b \text{ mod } p$$

here, $\mathcal{M} = \mathcal{T} = Z_p, \mathcal{K} = Z_p \times Z_p$.

Theorem: For any prime p , h is Strongly Universal.

Proof: Consider $m \neq m'$, and two outputs t, t' , $(am + b = t) - (am' + b = t') = (a(m - m') = (t - t')) \iff a = (t - t')(m - m')^{-1}$

$(m - m')^{-1}$ exists and is unique when $m \neq m'$ and so $\iff b = t - am = t - (t - t')(m - m')^{-1}m$.

These values of a and b define the unique $h_{a,b}$ such that $h_{a,b}(m) = t, h_{a,b}(m') = t'$. We thus have that

$$\forall m \neq m', t, t' \text{ s.t. } \Pr[h_k(m) = t \wedge h_k(m') = t'] = \frac{1}{|\mathcal{T}|^2}$$

In h , seeing two message-tag pairs completely determines a and b . Therefore, seeing two message-tag pairs enables to determine all further message-tag pairs.

Unfortunately, the one-time MAC authentication scheme has a number of drawbacks.

- The key is required to be twice as long as the message.
 - a rather long key must be securely stored,

- the error probability of $1/|\mathcal{M}|$ may be a serious overkill. We may be happy with, say, $1/2^{50}$ for all sizes of messages.

- As the name indicates — One-time MAC is only “secure” if keys are used only once.

Difference Universal Functions: Let \mathcal{T} be a group. A function $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ is ϵ -difference universal if for all distinct $m, m' \in \mathcal{M}$ and all $\Delta \in \mathcal{T}$ it holds that

$$\Pr[h_k(m) - h_k(m') = \Delta] \leq \epsilon$$

where the probability is taken over uniform choice of $k \in \mathcal{K}$.

CONSTRUCTION 4.24

Let $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ be a difference-universal function. Define a MAC for messages in \mathcal{M} as follows:

- **Gen:** choose uniform $k \in \mathcal{K}$ and $r \in \mathcal{T}$; output the key (k, r) .
- **Mac:** on input a key (k, r) and a message $m \in \mathcal{M}$, output the tag $t := h_k(m) + r$. (Addition here is done in the group \mathcal{T} .)
- **Vrfy:** on input a key (k, r) , a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$, output 1 if and only if $t \stackrel{?}{=} h_k(m) + r$.

Figure 87

Fix a constant l and let $\mathcal{M} = F^{<l}$, i.e., \mathcal{M} consists of vectors over F containing fewer than l entries. For any $m = (m_1, \dots, m_{l'-1}) \in \mathcal{M}$, where $l' \leq l$, let $m_{l'} \in F$ be an encoding of the length of m (i.e., $l' - 1$), and define the polynomial

$$m(X) = m_1 \cdot X^{l'} + m_2 \cdot X^{l'-1} + \dots + m_{l'} \cdot X$$

Finally, define the keyed function $h : F \times F^{<l} \rightarrow F$ as

$$h_k(m) = m(k)$$

Theorem: The function h above is $l/|F|$ -difference universal.

20 Lecture21

Constructing Secure MACs

CONSTRUCTION 4.5

Let F be a (length preserving) pseudorandom function. Define a fixed-length MAC for messages of length n as follows:

- **Mac**: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, output the tag $t := F_k(m)$.
- **Vrfy**: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^n$, and a tag $t \in \{0,1\}^n$, output 1 if and only if $t \stackrel{?}{=} F_k(m)$.

Figure 88

Extension to Variable-Length Messages**CONSTRUCTION 4.7**

Let $\Pi' = (\text{Mac}', \text{Vrfy}')$ be a fixed-length MAC for messages of length n . Define a MAC as follows:

- **Mac**: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of (nonzero) length $\ell < 2^{n/4}$, parse m into d blocks m_1, \dots, m_d , each of length $n/4$. (The final block is padded with 0s if necessary.) Choose a uniform identifier $r \in \{0,1\}^{n/4}$.
For $i = 1, \dots, d$, compute $t_i \leftarrow \text{Mac}'_k(r \parallel \ell \parallel i \parallel m_i)$, where i, ℓ are encoded as strings of length $n/4$.[†] Output the tag $t := \langle r, t_1, \dots, t_d \rangle$.
- **Vrfy**: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of length $\ell < 2^{n/4}$, and a tag $t = \langle r, t_1, \dots, t_{d'} \rangle$, parse m into d blocks m_1, \dots, m_d , each of length $n/4$. (The final block is padded with 0s if necessary.) Output 1 if and only if $d' = d$ and $\text{Vrfy}'_k(r \parallel \ell \parallel i \parallel m_i, t_i) = 1$ for $1 \leq i \leq d$.

[†] Note that i and ℓ can be encoded using $n/4$ bits because $i, \ell < 2^{n/4}$.

Figure 89

Theorem: If Π' is a secure fixed-length MAC for messages of length n , then Construction 4.7 is a MAC that is existentially unforgeable under an adaptive chosen-message attack (for arbitrary length up to $2^{n/4}$).

CBC-MAC

CONSTRUCTION 4.9

Let F be a pseudorandom function, and fix a length function $\ell > 0$. The basic CBC-MAC construction is as follows:

- **Mac**: on input a key $k \in \{0, 1\}^n$ and a message m of length $\ell(n) \cdot n$, do the following (we set $\ell = \ell(n)$ in what follows):
 1. Parse m as $m = m_1, \dots, m_\ell$ where each m_i is of length n .
 2. Set $t_0 := 0^n$. Then, for $i = 1$ to ℓ :
Set $t_i := F_k(t_{i-1} \oplus m_i)$.
 Output t_ℓ as the tag.
- **Vrfy**: on input a key $k \in \{0, 1\}^n$, a message m , and a tag t , do: If m is not of length $\ell(n) \cdot n$ then output 0. Otherwise, output 1 if and only if $t \stackrel{?}{=} \text{Mac}_k(m)$.

Figure 90

Theorem: Let l be a polynomial. If F is a pseudorandom function, then Construction 4.9 is a fixed-length MAC for messages of length $l(n) \cdot n$ that is existentially unforgeable under an adaptive chosen message attack.

- We stress that even though Construction 4.9 can be extended in the obvious way to handle messages whose length is an arbitrary multiple of n , this construction is only secure when the length of the messages being authenticated is fixed.
- That is, if an adversary is able to obtain MAC tags for messages of varying lengths, then the scheme is no longer secure.

There are two differences between the basic CBC-MAC and the CBC mode of encryption:

1. CBC-mode encryption uses a random IV and this is crucial for obtaining security. In contrast, CBC-MAC uses no IV (or the fixed value $IV = 0^n$) and this is also crucial for obtaining security. Specifically, CBC-MAC using a random IV is not secure.
2. In CBC-mode encryption all blocks $t_i(c_i)$ are output by the encryption algorithm as part of the ciphertext, whereas in CBC-MAC only the final block is output. This may seem to be a technical difference resulting from the fact that, for the case of encryption, all blocks must be output in order to enable decryption, whereas for a MAC this is simply not necessary and so is not done. However, if CBC-MAC is modified to output all blocks then it is no longer secure.

- In order to obtain a secure version of CBC-MAC for variable-length messages, Construction 4.9 must be modified.
- This can be done in a number of ways. Two possible options that can be proven secure are:
 1. Prepend the message with its length $||m||$ (encoded as an n -bit string), and then compute the basic CBC-MAC on the resulting message. (This is shown in Figure 4.2.) We stress that appending the block length to the end of the message is not secure.

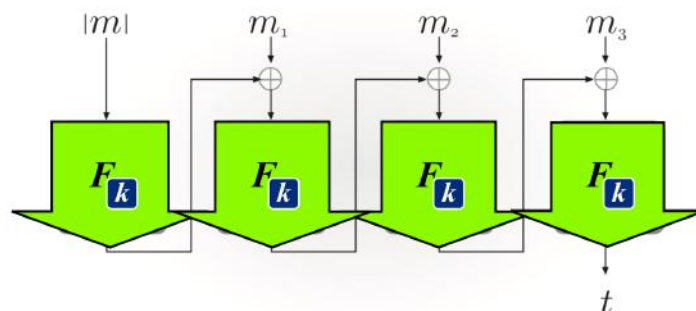


FIGURE 4.2: A variant of CBC-MAC secure for authenticating arbitrary-length messages.

Figure 91

2. Change the scheme so that key generation chooses two independent keys $k_1 \leftarrow \{0, 1\}^n$ and $k_2 \leftarrow \{0, 1\}^n$. Then, to authenticate a message m first compute the basic CBC-MAC of m using k_1 and let t be the result; output the tag $t' = F_{k_2}(t)$.

THEOREM 4.16 Let h be an $\varepsilon(n)$ -difference universal function for a negligible function ε , and let F be a pseudorandom function. Then Construction 4.15 is a strongly secure MAC for messages in $\{\mathcal{M}_n\}$.

CONSTRUCTION 4.15

Let h, F be as in the text. Define a MAC for messages in $\{\mathcal{M}_n\}$ as follows:

- **Gen:** on input 1^n , choose uniform $k_h \in \mathcal{K}_n$ and $k_F \in \{0, 1\}^n$; output the key (k_h, k_F) .
- **Mac:** on input a key (k_h, k_F) and a message $m \in \mathcal{M}_n$, choose a uniform $r \in \{0, 1\}^n$ and output the tag $t := \langle r, h_{k_h}(m) + F_{k_F}(r) \rangle$.
- **Vrfy:** on input a key (k_h, k_F) , a message $m \in \mathcal{M}_n$, and a tag $t = \langle r, s \rangle$, output 1 if and only if $s \stackrel{?}{=} h_{k_h}(m) + F_{k_F}(r)$.

Figure 92: A MAC based on a difference-universal function

- Suppose Alice and Bob have a secure MAC (Gen , Mac , $Vrfy$) and that Alice wants to send a secret bit b to Bob.
- She may pick two random messages m_0, m_1 , with a random tag $r \neq Mac_k(m_i), i \in \{0, 1\}$, and sends: $(m_0, Mac_k(m_0)), (m_1, r)$ to transmit $b = 0$ or $(m_0, r), (m_1, Mac_k(m_1))$ to transmit $b = 1$.
- Bob upon receiving two message-tag pairs $(m_0, t_0), (m_1, t_1)$ decrypts the bit b as $b = 0$ if $Vrfy_k(m_0, t_0) = 1$ and $b = 1$ if $Vrfy_k(m_1, t_1) = 1$.
- With probability one, Bob can tell which of the two messages is authenticated properly, whereas an adversary who does not know the key k cannot determine the validity of the two messages...

Privacy only vs. privacy and message integrity: It is best practice to always encrypt and authenticate by default.

Security requirements:

- The key-generation algorithm Gen' takes input 1^n , and runs $Gen_E(1^n)$ and $Gen_M(1^n)$ to obtain keys k_E and k_M respectively. The key is $k = (k_E, k_M)$.
- The Encryption $EncMac'$ takes as input the key k and a message m and outputs a value c that is derived by applying some combination of $Enc_{k_E}(\cdot)$ and $Mac_{k_M}(\cdot)$.
- The decryption algorithm Dec' takes as input the key k and a transmitted value c , and applies some combination of $Dec_{k_E}(\cdot)$ and $Vrfy_{k_M}(\cdot)$.
- The output of Dec' is either a plaintext m or a special symbol \perp that indicates an error (of authentication).
- The correctness requirement is that for every n , every pair of keys (k_E, k_M) output by $Gen'(1^n)$, and every value $m \in \{0, 1\}^*$,

$$Dec'_{k_E, k_M}(EncMac_{k_E, k_M}(m)) = m$$

Unforgeable-Encryption experiment $Enc - Forge_{A, \Pi'}(n)$:

1. A random key $k = (k_E, k_M)$ is generated by running $Gen'(1^n)$.

2. The adversary A is given input 1^n and oracle access to the algorithm $EncMac'_{k_E, k_M}(\cdot)$. The adversary eventually outputs c . Let \mathcal{Q} denote the set of all queries that A asked to its oracle.
3. Let $m = Dec'_{k_E, k_M}(c)$. The output of the experiment is 1 if and only if (1) $m \neq \perp$ and (2) $m \notin \mathcal{Q}$.

Security requirements: An Encryption scheme Π' is Unforgeable if for all PPT adversaries A , there exists a negligible function $negl$ such that

$$Pr[Enc - Forge_{A, \Pi'}(n) = 1] \leq negl(n)$$

Security requirements: A private-key encryption scheme is an authenticated encryption (AE) scheme if it is CCA-secure and unforgeable.

A common mistake is to use the same key for both encryption and authentication. This should never be done, as independent keys should always be used for independent applications (unless a specific proof of security when using the same key is known).

Obtaining Privacy and Authentication

1. **Encrypt-and-authenticate:** In this method, encryption and message authentication are computed independently. That is, given a plaintext message m , the sender transmits $\langle c, t \rangle$ where: $c \leftarrow Enc_{k_E}(m)$ and $t \leftarrow Mac_{k_M}(m)$. The receiver decrypts c to recover m , and then verifies the tag t . If $Vrfy_{k_M}(m, t) = 1$, the receiver outputs m ; otherwise it outputs \perp .
2. **Authenticate-then-encrypt:** Here a MAC tag t is first computed, and then the message and tag are encrypted together. That is, the sender transmits c computed as: $t \leftarrow Mac_{k_M}(m)$ and $c \leftarrow Enc_{k_E}(m || t)$. The authentication tag t is not sent “in the clear”, but is instead incorporated into the plaintext that is encrypted. The receiver decrypts c , and then verifies the tag t on m . As before, if $Vrfy_{k_M}(m, t) = 1$ the receiver outputs m ; otherwise, it outputs \perp .
3. **Encrypt-then-authenticate:** In this case, the message m is first encrypted and then a MAC tag is computed over the encrypted message. That is, the message is the pair $\langle c, t \rangle$ where: $c \leftarrow Enc_{k_E}(m)$ and $t \leftarrow Mac_{k_M}(c)$. The receiver verifies t before decrypting c . As before, if $Vrfy_{k_M}(c, t) = 1$ the receiver outputs m ; otherwise, it outputs \perp .

Encrypt-and-authenticate is not (necessarily) secure, since it may violate privacy. Authenticate-then-encrypt is also not necessarily secure.

CONSTRUCTION 5.6

Let $\Pi_E = (\text{Enc}, \text{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\text{Mac}, \text{Vrfy})$ be a message authentication code, where in each case key generation is done by simply choosing a uniform n -bit key. Define a private-key encryption scheme $(\text{Gen}', \text{Enc}', \text{Dec}')$ as follows:

- Gen' : on input 1^n , choose independent, uniform $k_E, k_M \in \{0, 1\}^n$ and output the key (k_E, k_M) .
- Enc' : on input a key (k_E, k_M) and a plaintext message m , compute $c \leftarrow \text{Enc}_{k_E}(m)$ and $t \leftarrow \text{Mac}_{k_M}(c)$. Output the ciphertext $\langle c, t \rangle$.
- Dec' : on input a key (k_E, k_M) and a ciphertext $\langle c, t \rangle$, first check whether $\text{Vrfy}_{k_M}(c, t) \stackrel{?}{=} 1$. If yes, then output $\text{Dec}_{k_E}(c)$; if no, then output \perp .

Figure 93: Encrypt-then-authenticate

Theorem: Let Π_E be a CPA-secure private-key encryption scheme, and let Π_M be a secure message authentication code (with unique tags). Then the combination $(\text{Gen}', \text{Mac}', \text{Vrfy}')$ derived by applying the encrypt-then-authenticate approach to Π_E, Π_M (Construction 5.6) is an Authenticated Encryption scheme.

The need for independent keys

- We conclude by stressing a basic principle of security and cryptography: different security goals should always use independent keys. That is, if an encryption scheme and a message authentication code are both needed, then independent keys should be used for each one.
- In order to illustrate this here, consider what can happen to the encrypt-then-authenticate methodology when the same key k is used for both encryption and authentication.
- Let F be a strong pseudorandom permutation. It follows that F^{-1} is also a strong pseudorandom permutations.
- Define $\text{Enc}_k(m) = F_k(m||r)$ for $m \in \{0, 1\}^{n/2}$ and a random $r \leftarrow \{0, 1\}^{n/2}$, and $\text{Mac}_k(c) = F_k^{-1}(c)$.

- It can be shown that the given encryption scheme is CPA-secure (in fact, it is even CCA-secure), and we know that the given message authentication code is a secure MAC.
- However, the encrypt-then-authenticate combination applied to the message m with the same key k yields:
- $Enc_k(m), Mac_k(Enc_k(m)) = F_k(m||r), F_k^{-1}(F_k(m||r)) = F_k(m||r), m||r$.

21 Lecture22

Chosen-Ciphertext Attacks Experiment $PrivK_{A,\Pi}^{cca}(n)$: Consider the following experiment for any private-key encryption scheme $\Pi = (Gen, Enc, Dec)$, adversary A , and value n for the security parameter. CCA indistinguishability experiment $PrivK_{A,\Pi}^{cca}(n)$:

1. A key k is generated by running $Gen(1^n)$
2. The adversary A is given input 1^n and oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$. It outputs a pair of messages m_0, m_1 of the same length.
3. A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a challenge ciphertext $c \leftarrow Enc_k(m_b)$ is computed and given to A .
4. The adversary A continues to have oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$, but is not allowed to query the latter on the challenge ciphertext itself. Finally, A outputs a bit b' .
5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Security Against CCA: A private-key encryption scheme Π has indistinguishable encryptions under a chosen-ciphertext attack (or is CCA-secure) if for all PPT adversaries A there exists a negligible function $negl$ such that

$$Pr[PrivK_{A,\Pi}^{cca}(n) = 1] \leq \frac{1}{2} + negl(n)$$

where the probability is taken over all random coins used in the experiment.

Theorem: If Π_E is a CPA-secure private-key encryption scheme, and Π_M is a secure MAC (with unique tags), then Construction 5.6 is a CCA-secure private-key encryption scheme.

Construction 5.6 may seem unsatisfying, since it achieves CCA-security simply by ensuring that the decryption oracle is useless to the adversary. Instead of being viewed as a drawback of the construction, this should be viewed as an advantage! Although there are other ways to achieve CCA-security, here the adversary is unable to generate any valid ciphertext that was not already created by one of the honest parties. This means that Construction 5.6 achieves both privacy and message authentication.

Authenticated Encryption vs. CCA-security

- Although we use the same construction for achieving CCA-security and Authenticated Encryption, the security goals in each case are different.
- In the setting of CCA-security we are not necessarily interested in obtaining message authentication; rather, we wish to ensure privacy even against a strong adversary who is able to make decryption queries.
- When considering Authenticated Encryption, in contrast, we are interested in the twin goals of CCA-security and integrity.
- Clearly, as we have defined it, Authenticated Encryption implies CCA-security. The opposite direction is not necessarily true.

To model this formally, we consider a modified experiment **Mac-sforge** that is defined in exactly the same way as **Mac-forge**, except that now the set \mathcal{Q} contains *pairs* of oracle queries and their associated responses. (That is, $(m, t) \in \mathcal{Q}$ if \mathcal{A} queried $\text{Mac}_k(m)$ and received in response the tag t .) The adversary \mathcal{A} succeeds (and experiment **Mac-sforge** evaluates to 1) if and only if \mathcal{A} outputs (m, t) such that $\text{Vrfy}_k(m, t) = 1$ and $(m, t) \notin \mathcal{Q}$.

DEFINITION 4.3 A message authentication code $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ is **strongly** secure if for all probabilistic polynomial-time adversaries \mathcal{A} , there is a negligible function negl such that:

$$\Pr[\text{Mac-sforge}_{\mathcal{A}, \Pi}(n) = 1] \leq \text{negl}(n).$$

Figure 94

Digital Signature Schemes Digital signature schemes allow a signer S who has established a public key pk , to “sign” a message (using a secret key sk) in such a way that any other party who knows pk (and knows that this public key was established by S) can “verify”

that the message originated from S and has not been modified in any way. A qualitative advantage that digital signatures have as compared to message authentication codes is that signatures are publicly verifiable. This means that if a receiver verifies the signature on a given message as being legitimate, then it is assured that all other parties who receive this signed message will also verify it as legitimate.

- **Public Verifiability:** Public verifiability implies that signatures are transferable : a signature σ on a message m by a particular signer S can be shown to a third party, who can then verify herself that σ is a legitimate signature on m with respect to S's public key.
- **Transferability:** By making a copy of the signature, this third party can then show the signature to another party and convince them that S authenticated m , and so on.
- **non-repudiation:** Digital signature schemes also provide the very important property of non-repudiation. That is — assuming a signer S widely publicizes his public key in the first place — once S signs a message he cannot later deny having done so.

Definitions of DSA:

DEFINITION 13.1. A signature scheme is a tuple of three probabilistic polynomial-time algorithms $(\text{Gen}, \text{Sign}, \text{Vrfy})$ satisfying the following:

1. The key-generation algorithm Gen takes as input a security parameter 1^n and outputs a pair of keys (pk, sk) . These are called the public key and the private key, respectively. We assume for convenience that pk and sk each have length at least n , and that n can be determined from pk, sk .
2. The signing algorithm Sign takes as input a private key sk and a message² $m \in \{0, 1\}^*$. It outputs a signature σ , denoted as $\sigma \leftarrow \text{Sign}_{sk}(m)$.
3. The deterministic verification algorithm Vrfy takes as input a public key pk , a message m , and a signature σ . It outputs a bit b , with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this as $b := \text{Vrfy}_{pk}(m, \sigma)$.

Figure 95

It is required that for every n , every (pk, sk) output by $\text{Gen}(1^n)$, and every $m \in \{0, 1\}^*$, it holds that

$$\text{Vrfy}_{pk}(m, \text{Sign}_{sk}(m)) = 1$$

If $(Gen, Sign, Vrfy)$ is such that for every (pk, sk) output by $Gen(1^n)$, algorithm $Sign_{sk}$ is only defined for messages $m \in \{0, 1\}^{l(n)}$, then we say that $(Gen, Sign, Vrfy)$ is a signature scheme for messages of length $l(n)$.

- A signature scheme is used in the following way. One party S, who acts as the sender, runs $Gen(1^n)$ to obtain keys (pk, sk) .
- The public key pk is then publicized as belonging to S; e.g., S can put the public key on its webpage or place it in some public directory.
- As in the case of public-key encryption, we assume that any other party is able to obtain a legitimate copy of S's public key.
- When S wants to transmit a message m , it computes the signature $\sigma \leftarrow Sign_{sk}(m)$ and sends (m, σ) .
- Upon receipt of (m, σ) , a receiver who knows pk can verify the authenticity of m by checking whether $Vrfy_{pk}(m, \sigma) = 1$.
- This establishes both that S sent m , and also that m was not modified in transit. As in the case of message authentication codes, however, it does not say anything about when m was sent, and replay attacks are still possible.

Security of signature schemes experiment $Sig - Forge_{A, \Pi}(n)$:

1. $Gen(1^n)$ is run to obtain keys (pk, sk) .
2. Adversary A is given pk and oracle access to $Sign_{sk}(\cdot)$. The adversary then outputs (m, σ) . Let \mathcal{Q} denote the set of messages whose signatures were requested by A during its execution.
3. The output of the experiment is defined to be 1 if and only if (i) $Vrfy_{pk}(m, \sigma) = 1$, and (ii) $m \notin \mathcal{Q}$.

Security of signature schemes: A signature scheme $\Pi = (Gen, Sign, Vrfy)$ is existentially unforgeable under an adaptive chosen-message attack if for all probabilistic polynomial-time adversaries A, there exists a negligible function $negl(n)$ such that

$$Pr[Sig - Forge_{A, \Pi}(n) = 1] \leq negl(n)$$

CONSTRUCTION 13.5

Let GenRSA be as in the text. Define a signature scheme as follows:

- **Gen**: on input 1^n run GenRSA(1^n) to obtain (N, e, d) . The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- **Sign**: on input a private key $sk = \langle N, d \rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the signature

$$\sigma := [m^d \bmod N].$$

- **Vrfy**: on input a public key $pk = \langle N, e \rangle$, a message $m \in \mathbb{Z}_N^*$, and a signature $\sigma \in \mathbb{Z}_N^*$, output 1 if and only if

$$m \stackrel{?}{=} [\sigma^e \bmod N].$$

Figure 96

A no-message Attack:

- It is trivial to output a forgery for the plain RSA signature scheme based on the public key alone, without even obtaining any signatures from the legitimate signer.
- Given a public key $pk = \langle N, e \rangle$, choose an arbitrary $\sigma \in \mathbb{Z}_N^*$ and compute $m = \sigma^e \bmod N$.
- Then output the forgery (m, σ) . It is immediate that σ is a valid signature on m , and this is obviously a forgery since no signature on m was generated by the owner of the public key.
- Say the adversary wants to forge a signature on the message $m \in \mathbb{Z}_N^*$ with respect to the public-key $pk = \langle N, e \rangle$.
- The adversary chooses a random $m_1 \in \mathbb{Z}_N^*$, sets $m_2 = m/m_1 \bmod N$, and then obtains signatures σ_1 and σ_2 on m_1 and m_2 respectively. We claim that $\sigma = \sigma_1 \cdot \sigma_2 \bmod N$ is a valid signature on m .
- This is because $\sigma^e \equiv (\sigma_1 \sigma_2)^e \equiv (m_1^d m_2^d)^e \equiv m_1^{ed} m_2^{ed} \equiv m_1 m_2 \equiv m \bmod N$ using the fact that σ_1, σ_2 are valid signatures of m_1, m_2 .

RSA-FDH

CONSTRUCTION 13.6

Let GenRSA be as in the previous sections, and construct a signature scheme as follows:

- **Gen**: on input 1^n , run GenRSA(1^n) to compute (N, e, d) . The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
As part of key generation, a function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$ is specified, but we leave this implicit.

- **Sign**: on input a private key $\langle N, d \rangle$ and a message $m \in \{0, 1\}^*$, compute

$$\sigma := [H(m)^d \bmod N].$$

- **Vrfy**: on input a public key $\langle N, e \rangle$, a message m , and a signature σ , output 1 if and only if $\sigma^e \stackrel{?}{=} H(m) \bmod N$.

Figure 97: The RSA-FDH signature scheme

An immediate observation is that a minimal requirement for the above scheme to be secure is that H must be collision-resistant. Since H must be a collision-resistant hash function, this modified scheme described is sometimes called the hashed RSA signature scheme or RSA full-domain hash (RSA-FDH).

Theorem: If the RSA problem is hard relative to GenRSA and H is modeled as a random oracle, then Construction 13.6 is secure.

- The no-message attack. The natural way to attempt the no-message attack shown previously is to choose an arbitrary $\sigma \in \mathbb{Z}_N^*$, compute $m' = \sigma^e \bmod N$, and then try to find some $m \in \{0, 1\}^*$ such that $H(m) = m'$.
- If the function H is not efficiently invertible this appears difficult to do.
- Forging a signature on an arbitrary message. The natural way to attempt the chosen-message attack shown previously requires the adversary to find three messages m, m_1, m_2 for which $H(m) = H(m_1) \cdot H(m_2) \bmod N$.
- Once again, if H is not efficiently invertible this seems difficult to do.

22 Lecture23

Hash Functions and Applications

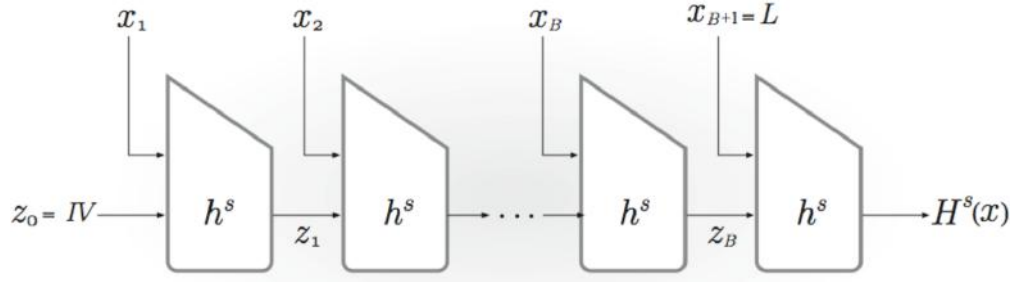


Figure 98: Collision-Resistant Hash Functions

Hash Functions

DEFINITION 6.1 A hash function (with output length ℓ) is a pair of probabilistic polynomial-time algorithms (Gen, H) satisfying the following:

- Gen is a probabilistic algorithm which takes as input a security parameter 1^n and outputs a key s . We assume that 1^n is implicit in s .
- H takes as input a key s and a string $x \in \{0, 1\}^*$ and outputs a string $H^s(x) \in \{0, 1\}^{\ell(n)}$ (where n is the value of the security parameter implicit in s).

If H^s is defined only for inputs $x \in \{0, 1\}^{\ell'(n)}$ and $\ell'(n) > \ell(n)$, then we say that (Gen, H) is a fixed-length hash function for inputs of length ℓ' . In this case, we also call H a compression function.

Figure 99

The collision-finding experiment $\text{Hash} - \text{coll}_{A, \Pi}(n)$:

1. A key s is generated by running $\text{Gen}(1^n)$.
2. The adversary A is given s and outputs x, x' . (If Π is a fixed-length hash function for inputs of length $\ell'(n)$ then we require $x, x' \in \{0, 1\}^{\ell'(n)}$)
3. The output of the experiment is defined to be 1 if and only if $x \neq x'$ and $H^s(x) = H^s(x')$. In such a case we say that A has found a collision.

Collision Resistant: A hash function $\Pi = (\text{Gen}, H)$ is collision resistant if for all probabilistic polynomial-time adversaries A there exists a negligible function negl such that

$$\Pr[\text{Hash} - \text{coll}_{A, \Pi}(n) = 1] \leq \text{negl}(n)$$

Weaker Notions of Secure Hash Functions

1. **Collision resistance:** This is the strongest notion and the one we have considered so far.
2. **Second pre-image resistance:** Informally speaking, a hash function is second pre-image resistant if given s and x it is infeasible for a PPT adversary to find $x' \neq x$ such that $H^s(x) = H^s(x')$.
3. **Pre-image resistance:** Informally, a hash function is pre-image resistant if given s and $y = H^s(x)$ (but not x) for a randomly-chosen x it is infeasible for a PPT adversary to find a value x' such that $H^s(x') = y$.

Generic “Birthday” Attack: Assume we are given a hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^l$. The attack works as follows: Choose q arbitrary distinct inputs $x_1, \dots, x_q \in \{0, 1\}^{2l}$, compute $y_i = H(x_i)$, and check whether any of the two y_i values are equal. What is the probability that this algorithm finds a collision? When q is $\Theta(2^{l/2})$, the probability of such a collision is roughly $1/2$. In the case of birthdays, it turns out that if there are 23 people in a room, the probability that two have the same birthday is greater than $1/2$. As an example, assume a hash function is designed with output length of 128 bits. It is clearly infeasible to run 2^{128} steps in order to find a collision. However, running for 2^{64} steps is within the realm of feasibility (though still rather difficult). Thus, the existence of generic birthday attacks mandates that any collision-resistant hash function in practice needs to have output that is longer than 128 bits.

ALGORITHM 6.9
A small-space birthday attack

Input: A hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$
Output: Distinct x, x' with $H(x) = H(x')$

```

 $x_0 \leftarrow \{0, 1\}^{\ell+1}$ 
 $x' := x := x_0$ 
for  $i = 1, 2, \dots$  do:
     $x := H(x)$ 
     $x' := H(H(x'))$ 
    // now  $x = H^{(i)}(x_0)$  and  $x' = H^{(2i)}(x_0)$ 
    if  $x = x'$  break
 $x' := x, x := x_0$ 
for  $j = 1$  to  $i$ :
    if  $H(x) = H(x')$  return  $x, x'$  and halt
    else  $x := H(x), x' := H(x')$ 
    // now  $x = H^{(j)}(x_0)$  and  $x' = H^{(i+j)}(x_0)$ 

```

Figure 100

The Merkle-Damgard Transform

CONSTRUCTION 6.3

Let (Gen, h) be a fixed-length hash function for inputs of length $2n$ and with output length n . Construct hash function (Gen, H) as follows:

- Gen : remains unchanged.
- H : on input a key s and a string $x \in \{0, 1\}^*$ of length $L < 2^n$, do the following:
 1. Set $B := \lceil \frac{L}{n} \rceil$ (i.e., the number of blocks in x). Pad x with zeros so its length is a multiple of n . Parse the padded result as the sequence of n -bit blocks x_1, \dots, x_B . Set $x_{B+1} := L$, where L is encoded as an n -bit string.
 2. Set $z_0 := 0^n$. (This is also called the IV.)
 3. For $i = 1, \dots, B + 1$, compute $z_i := h^s(z_{i-1} || x_i)$.
 4. Output z_{B+1} .

Figure 101: Also see Figure. 98

The security of the Merkle-Damgard Transform: The intuition behind the security of the Merkle Damgard transform is that if two different strings x and x' collide in H^s , then there must be distinct intermediate values $z_{i-1} || x_i$ and $z'_{i-1} || x'_i$ in the computation of $H^s(x)$ and $H^s(x')$ respectively such that $h^s(z_{i-1} || x_i) = h^s(z'_{i-1} || x'_i)$

Theorem: If (Gen, h) is a fixed-length collision-resistant hash function, then (Gen, H) is a collision-resistant hash function.

Collision-Resistant Hash Functions in Practice:

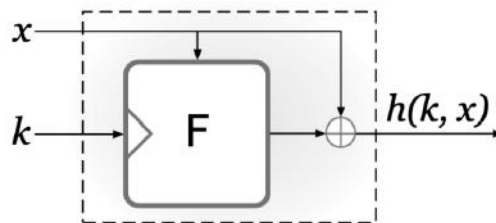


Figure 102: The Davies-Meyer Construction

THEOREM 6.5 *If F is modeled as an ideal cipher, then the Davies–Meyer construction yields a collision-resistant compression function. Concretely, any attacker making $q < 2^{\ell/2}$ queries to its ideal-cipher oracles finds a collision with probability at most $q^2/2^\ell$.*

Figure 103

The Davies–Meyer construction is a useful paradigm for constructing collision-resistant com-

pression functions. However, it should not be applied to block ciphers designed for encryption, like DES and AES. The ideal-cipher model is a strengthening of the random-oracle model, in which we posit that all parties have access to an oracle for a random keyed permutation $F : \{0, 1\}^n \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ as well as its inverse F^{-1} (i.e., such that $F^{-1}(k, F(k, x)) = x$ for all k, x). Another way to think of this is that each key $k \in \{0, 1\}^n$ specifies an independent, uniform permutation $F(k, \cdot)$ on l -bit strings. As in the random-oracle model, the only way to compute F (or F^{-1}) is to explicitly query the oracle with (k, x) and receive back $F(k, x)$ (or $F^{-1}(k, x)$).

The Sponge Construct

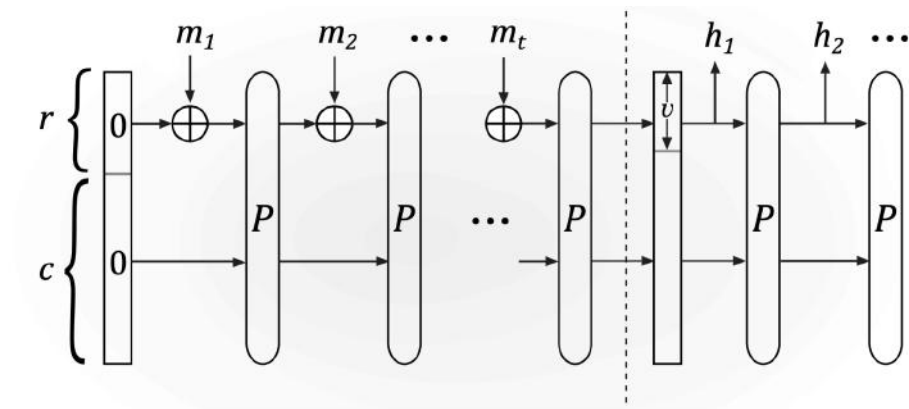


Figure 104: The sponge construction. The absorbing phase is to the left of the dashed line, and the squeezing phase is to the right.

CONSTRUCTION 7.6

Fix $P : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$ and constants r, c, v as in the text and $\lambda \geq 1$. Hash function H , on input $\hat{m} \in \{0, 1\}^*$, does:

(Padding) Append a 1 to \hat{m} , followed by enough zeros so that the length of the resulting string is a multiple of r . Parse the resulting string as the sequence of r -bit blocks m_1, \dots, m_t .

(Absorbing phase) Set $y_0 := 0^\ell$. Then for $i = 1, \dots, t$ do:

- $x_i := y_{i-1} \oplus (m_i \| 0^c)$.
- $y_i := P(x_i)$.

(Squeezing phase) Set $y_1^* := y_t$, and let h_1 be the first v bits of y_1^* . Then for $i = 2, \dots, \lambda$ do

- $y_i^* := P(y_{i-1}^*)$.
- Let h_i be the first v bits of y_i^* .

(Output) Output $h_1 \| \dots \| h_\lambda$.

Figure 105: A hash function based on the sponge construction.

The “Hash-and-Sign” Paradigm

CONSTRUCTION 13.3

Let $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ be a signature scheme for messages of length $\ell(n)$, and let $\Pi_H = (\text{Gen}_H, H)$ be a hash function with output length $\ell(n)$. Construct signature scheme $\Pi' = (\text{Gen}', \text{Sign}', \text{Vrfy}')$ as follows:

- **Gen'**: on input 1^n , run $\text{Gen}(1^n)$ to obtain (pk, sk) and run $\text{Gen}_H(1^n)$ to obtain s ; the public key is $\langle pk, s \rangle$ and the private key is $\langle sk, s \rangle$.
- **Sign'**: on input a private key $\langle sk, s \rangle$ and a message $m \in \{0, 1\}^*$, output $\sigma \leftarrow \text{Sign}_{sk}(H^s(m))$.
- **Vrfy'**: on input a public key $\langle pk, s \rangle$, a message $m \in \{0, 1\}^*$, and a signature σ , output 1 if and only if $\text{Vrfy}_{pk}(H^s(m), \sigma) \stackrel{?}{=} 1$.

Figure 106: The hash-and-sign paradigm

THEOREM 13.4 *If Π is a secure signature scheme for messages of length ℓ and Π_H is collision resistant, then Construction 13.3 is a secure signature scheme (for arbitrary-length messages).*

Figure 107

The Digital Signature Algorithm

CONSTRUCTION 12.13

Let \mathcal{G} be as in the text.

- **Gen:** on input 1^n , run $\mathcal{G}(1^n)$ to obtain (\mathbb{G}, q, g) . Choose uniform $x \in \mathbb{Z}_q$ and set $y := g^x$. The public key is $\langle \mathbb{G}, q, g, y \rangle$ and the private key is x .

As part of key generation, two functions $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ and $F : \mathbb{G} \rightarrow \mathbb{Z}_q$ are specified, but we leave this implicit.

- **Sign:** on input the private key x and a message $m \in \{0, 1\}^*$, choose uniform $k \in \mathbb{Z}_q^*$ and set $r := F(g^k)$. Then compute $s := [k^{-1} \cdot (H(m) + xr) \bmod q]$. (If $r = 0$ or $s = 0$ then start again with a fresh choice of k .) Output the signature (r, s) .
- **Vrfy:** on input a public key $\langle \mathbb{G}, q, g, y \rangle$, a message $m \in \{0, 1\}^*$, and a signature (r, s) with $r, s \neq 0 \bmod q$, output 1 if and only if

$$r \stackrel{?}{=} F\left(g^{H(m) \cdot s^{-1}} y^{r \cdot s^{-1}}\right).$$

Figure 108: DSA and ECDSA abstractly

CONSTRUCTION 12.15 (Katz-Lindell 1st Ed.)

Let \mathcal{G} be as in the text. Define a signature scheme as follows:

- **Gen:** on input 1^n , run the algorithm $\mathcal{G}(1^n)$ to obtain (p, q, g) . Let $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ be function. Choose $x \leftarrow \mathbb{Z}_q$ uniformly at random and set $y := [g^x \bmod p]$. The public key is $\langle H, p, q, g, y \rangle$ and the private key is $\langle H, p, q, g, x \rangle$.
- **Sign:** on input a private key $\langle H, p, q, g, x \rangle$ and a message $m \in \{0, 1\}^*$, choose $k \leftarrow \mathbb{Z}_q^*$ uniformly at random and set $r := [[g^k \bmod p] \bmod q]$. Compute $s := [(H(m) + xr) \cdot k^{-1} \bmod q]$, and output the signature (r, s) .
- **Vrfy:** on input a public key $\langle H, p, q, g, y \rangle$, a message $m \in \{0, 1\}^*$, and a signature (r, s) with $r \in \mathbb{Z}_q$ and $s \in \mathbb{Z}_q^*$, compute the values $u_1 := [H(m) \cdot s^{-1} \bmod q]$ and $u_2 := [r \cdot s^{-1} \bmod q]$. Output 1 if and only if

$$r \stackrel{?}{=} [[g^{u_1} y^{u_2} \bmod p] \bmod q].$$

Figure 109: The Digital Signature Standard (DSS)

- Let \mathcal{G} be a PPT algorithm that, on input 1^n , outputs (p, q, g) where, except with negligible probability:

1. p and q are primes with $||q|| = n$

2. $q|(p-1)$ but $q^2 \nmid (p-1)$

3. g is a generator of the order q subgroup of Z_p^*

- Let us see that the scheme is correct. Letting $m' = H(m)$, the signature (r, s) output by the signer satisfies

$$r = [g^k \bmod p] \bmod q$$

$$s = (m' + xr) \cdot k^{-1} \bmod q$$

and we check

$$r = [g^{u_1} y^{u_2} \bmod p] \bmod q$$

with $u_1 = m' \cdot s^{-1}$ and $u_2 = r \cdot s^{-1}$

- Assume $s \neq 0$ (The opposite occurs with only negligible probability). Using the fact that $y = g^x$ and recalling that we can work "in the exponent" modulo q , we have

$$\begin{aligned} g^{u_1} y^{u_2} &= g^{m' s^{-1}} (g^x)^{r s^{-1}} \\ &\equiv g^{m' (m' + xr)^{-1} k} g^{xr (m' + xr)^{-1} k} \pmod{p} \\ &\equiv g^{(m' + xr) (m' + xr)^{-1} k} \equiv g^k \pmod{p} \end{aligned}$$

Thus

$$[g^{u_1} y^{u_2} \bmod p] \bmod q = [g^k \bmod p] \bmod q = r$$

and verification succeeds.

23 Lecture24

Certificates and P-K Infrastructures

- The key idea is the notion of a digital certificate, which is simply a signature binding some entity to some public key.
- To be concrete, say a party Charlie has generated a key-pair (pk_C, sk_C) for a secure digital signature scheme
- Assume further that another party Bob has also generated a key-pair (pk_B, sk_B) (in the present discussion, these may be keys for either a signature scheme or a public-key encryption scheme), and that Charlie knows that pk_B is Bob's public key.

- Then Charlie can compute the signature

$$cert_{C \rightarrow B} = Sign_{sk_C}('Bob's' key is pk'_B)$$

and give this signature to Bob.

- This signature $cert_{C \rightarrow B}$ is called a certificate for Bob's key issued by Charlie.
- In practice a certificate should unambiguously identify the party holding a particular public key and so a more uniquely descriptive term than "Bob" would be used, for example, Bob's full name and email address.
- Now say Bob wants to communicate with some other party Alice who already knows pk_C
- What Bob can do is to send $(pk_B, cert_{C \rightarrow B})$ to Alice, who can then verify the validity of the signature on the message 'Bob's key is pk_B ' with respect to pk_C
- Assuming verification succeeds, Alice now knows that Charlie has signed the indicated message.
- If Alice trusts Charlie, then she might now accept pk_B as Bob's legitimate public key.
- Note that all communication between Bob and Alice can occur over an insecure and unauthenticated channel.
- If an active adversary interferes with the communication of $(pk_B, cert_{C \rightarrow B})$ from Bob to Alice, that adversary will be unable to generate a valid certificate linking Bob to any other public key pk'_B unless Charlie had previously done so.
- This all assumes that Charlie is not dishonest and that his private signing key has not been compromised.

A single certificate authority

- The simplest PKI assumes a single certificate authority (CA) who is completely trusted by everybody and who issues certificates for everyone's public key.
- A certificate authority would not typically be a person, but would more likely be a company whose business it is to certify public keys, a governmental agency, or perhaps a department within an organization.

- Anyone who wants to rely on the services of the CA would have to obtain a legitimate copy of the CA's public key pk_{CA}

Multiple certificate authorities

- Outside of a single organization it is highly unlikely for everyone to trust the same CA.
- This need not imply that anyone thinks the CA is corrupt; it could simply be the case that someone finds the CA's verification process to be insufficient.
- Moreover, the CA is a single point of failure for the entire system. If the CA is corrupt, or can be bribed, or even if the CA is merely lax with the way it protects its private signing key, the legitimacy of issued certificates may be called into question.
- Reliance on a single CA is also a problem even in non-adversarial environments:
 - if the CA is unreachable then no new certificates can be issued
 - the load on the CA may be very high if many parties want to obtain certificates at once (although this is still better than the KDC model because the CA does not need to be online once certificates are issued).
- A party Bob who wants to obtain a certificate on his public key can choose which CA(s) it wants to issue a certificate, and a party Alice who is presented with a certificate, or even multiple certificates issued by different CAs, can choose which CA's certificates she trusts.
- There is no harm in having Bob obtain a certificate from every CA (apart from some inconvenience and expense for Bob), but Alice must be more careful since the security of her communications is ultimately only as good as the least-secure CA that she trusts.

Delegation and certificate chains

- Say Charlie, acting as a CA, issues a certificate for Bob as in our original discussion.
- Bob, in turn, can issue his own certificates for other parties. For example, Bob may issue a certificate for Alice of the form

$$cert_{B \rightarrow A} = \text{Sign}_{sk_B}('Alice's\ key\ is\ pk'_A)$$

- Now, if Alice wants to communicate with some fourth party Dave who knows Charlie's public key (but not Bob's), then Alice can send

$$pk_A, cert_{B \rightarrow A}, pk_B, cert_{C \rightarrow B}$$

to Dave.

- Dave can first verify that Charlie, whom he trusts and whose public key is already in his possession, has signed a certificate $cert_{C \rightarrow B}$ indicating that pk_B indeed belongs to someone named Bob.
- Dave can also verify that this person named Bob has signed a certificate $cert_{B \rightarrow A}$ indicating that pk_A indeed belongs to Alice.
- If Dave trusts Charlie to only issue certificates to trustworthy people, then Dave may accept pk_A as being the authentic key of Alice.

The “web of trust” model

- In the “web of trust” model, anyone can issue certificates to anyone else and each user has to make their own decision about how much trust to place in certificates issued by other users.
- As an example of how this might work, say a user Alice is already in possession of public keys pk_1, pk_2, pk_3 for some users C_1, C_2, C_3 .
- Another user Bob who wants to communicate with Alice might have certificates $cert_{C_1 \rightarrow B}, cert_{C_3 \rightarrow B}$, and $cert_{C_4 \rightarrow B}$ and will send these certificates (along with his public key pk_B to Alice.
- Alice cannot verify $cert_{C_4 \rightarrow B}$ (she doesn't have C_4 's public key), but she can verify the other two certificates.
- Now she has to decide how much she trusts C_1 and C_3
- She may decide to accept pk_B if she unequivocally trusts C_1 , or if she trusts both C_1 and C_3 to a lesser extent.
- (She may, for example, consider it likely that either C_1 or C_3 is corrupt, but consider it unlikely for them both to be corrupt.)

Invalidating Certificates: Expiration

- One method for preventing certificates from being used indefinitely is to include an expiry date as part of the certificate.
- A certificate issued by a CA Charlie for Bob's public key might now have the form

$$cert_{C \rightarrow B} = \text{Sign}_{sk_C}('Bob's\ key\ is\ pk'_B, \text{ date})$$

where “date” is some date in the future at which point the certificate becomes invalid. (For example, it may be one year from the day the certificate is issued.)

- When another user verifies this certificate, they need to know not only pk_B but also the expiry date, and they now need to check not only that the signature is valid, but also that the expiry date has not passed.
- A user who holds a certificate must contact the CA to get a new certificate issued whenever their current one expires; at this point, the CA verifies the identity/credentials of the user again before issuing another certificate.

Invalidating Certificates:Revocation

- When an employee leaves an organization, or a user's private key is stolen, we would like the certificates that have been issued for their public keys to become invalid immediately, or at least as soon as possible.
- This can be achieved by having the CA explicitly revoke the certificate. Of course, everything we say applies more generally if the user had certificates issued by multiple CAs; for simplicity we assume a single CA.
- There are many different ways revocation can be handled. One possibility is for the CA to include a serial number in every certificate it issues; that is, a certificate will now have the form

$$cert_{C \rightarrow B} = \text{Sign}_{sk_C}('Bob's\ key\ is\ pk'_B, \text{ #####})$$

where '#####' represents its serial number.

- Each certificate should have a unique serial number, and the CA will store the information $(Bob, pk_B, #####)$ for each certificate it generates.
- If a user Bob's private key corresponding to the public key pk_B is stolen, then Bob can alert the CA.

- The CA will then search its database to find the serial number of the certificate issued for Bob and pk_B .
- At the end of each day, say, the CA will generate a certificate revocation list (or CRL) containing the serial numbers of all revoked certificates, and sign this entire list along with the current date. The signed list is then widely distributed, perhaps by posting it on the CA's public webpage.
- To verify a certificate issued as above, another user now needs pk_B and also the serial number of the certificate (this can be forwarded by Bob along with everything else).
- Verification now requires checking that the signature is valid, checking that the serial number does not appear on the most recent revocation list, and verifying the CA's signature on the revocation list itself.

Putting It All Together – SSL/TLS

- TLS is a standardized protocol based on a precursor called SSL (or Secure Socket Layer) that was developed by Netscape in the mid-1990s; the last version available was SSL 3.0.
- TLS version 1.0 was released in 1999, updated to version 1.1 in 2006, and updated again to version 1.2 in 2008, and finally, version 1.3 (the current version) in 2018.
- All major web browsers support TLS 1.3, although in some cases earlier versions of TLS are used by default.
- For the most part, our description below is at a sufficiently high level that the differences between the versions are unimportant for our purposes.
- We caution, however, that there are several known attacks on earlier versions.
- The TLS protocol allows a client (e.g., a web browser) and a server (e.g., a website) to agree on a set of shared keys and then use those keys to encrypt and authenticate their subsequent communication.
- It consists of two parts: a handshake protocol that performs (authenticated) key exchange to establish the shared keys, and a record-layer protocol that uses those shared keys to encrypt/ authenticate the parties' communication.

TLS Handshake

- The handshake protocol. We describe the basic flow of the protocol in the most typical case.
 - At the outset, the client C holds a set of CAs' public keys $\{pk_1, \dots, pk_n\}$, and the server S holds keys (pk_S, sk_S) for a digital signature scheme along with a certificate $cert_{i \rightarrow S}$ on pk_S issued by one of the CAs whose public key C knows.
 - The parties run the following steps.
1. C begins by sending to S the initial message of the Diffie–Hellman key-exchange protocol. This message includes (\mathcal{G}, q, g, g^x) . The underlying group is selected by the client from a set of standardized options, and can be either a prime-order subgroup of Z_p^* for some prime p or an elliptic-curve group. The client also sends a uniform value (a “nonce”) $NC \in \{0, 1\}^n$. This message from C also includes information about which cryptographic algorithms (or cipher-suites) are supported by the client.
 2. S completes the Diffie–Hellman key exchange by sending a message to the client containing g^y for a random secret value y chosen by the server. The server also includes its own uniform value $NS \in \{0, 1\}^n$. At this point, S can compute a shared secret $K = g^{xy}$. It applies a key derivation function to K to derive keys k'_S, k'_C, k_S, k_C for an authenticated encryption (AE) scheme. Supported AE schemes include GCM, CCM, and ChaCha20–Poly1305. Finally, S sends its public key pk_S and its certificate $cert_{i \rightarrow S}$, along with a signature σ computed by the server (using its long-term key sk_S) on the handshake messages exchanged thus far. These values sent by the server are all encrypted using k'_S .
 3. C computes K from the server's response, and also derives the keys k'_S, k'_C, k_S, k_C . It uses k'_S to recover pk_S and the associated certificate, as well as the signature σ . The client checks whether one of the CA's public keys that it holds matches the CA who issued S's certificate. If so, C verifies the certificate (and also checks that it has not expired or been revoked) and, if this was successful, learns that pk_S is indeed S's public key. C then verifies the signature σ on the handshake messages with respect to pk_S , and aborts if verification fails. Finally, C computes a MAC of the handshake messages exchanged thus far using k'_C . It sends the result back to S, who verifies the tag before proceeding to the record-layer protocol.

TLS Handshake security(intuition):

- Since C verifies the certificate, then pk_S is the public key of S.
- Since signature σ is valid, then C knows it is communicating with S. (It is important here that the handshake messages being signed have high entropy, so as to prevent a replay attack. This is why the client includes a random nonce NC as part of its initial message.)
- Since S signs all the messages of the DH key-exchange protocol, C knows that none of those values were tampered with as would be the case if an active adversary were carrying out a man-in-the-middle attack.
- Of course, the Diffie–Hellman protocol itself ensures that a passive eavesdropper learns nothing about K (and derived keys).

TLS Record-layer: The record-layer protocol. Once keys have been agreed upon by C and S, the parties use those keys to encrypt and authenticate all their subsequent communication using an AE scheme. C uses k_C for the messages it sends to S, whereas S uses k_S for the messages it sends to C. Sequence numbers are used to prevent replay attacks.

Lamport's One-Time Signature Scheme

Signing $m = 011$:

$$sk = \left(\begin{array}{|c|c|c|} \hline x_{1,0} & x_{2,0} & x_{3,0} \\ \hline x_{1,1} & x_{2,1} & x_{3,1} \\ \hline \end{array} \right) \Rightarrow \sigma = (x_{1,0}, x_{2,1}, x_{3,1})$$

Verifying for $m = 011$ and $\sigma = (x_1, x_2, x_3)$:

$$pk = \left(\begin{array}{|c|c|c|} \hline y_{1,0} & y_{2,0} & y_{3,0} \\ \hline y_{1,1} & y_{2,1} & y_{3,1} \\ \hline \end{array} \right) \left. \vphantom{\begin{array}{|c|c|c|} \hline y_{1,0} & y_{2,0} & y_{3,0} \\ \hline y_{1,1} & y_{2,1} & y_{3,1} \\ \hline \end{array}} \right\} \Rightarrow \begin{array}{l} f(x_1) \stackrel{?}{=} y_{1,0} \\ f(x_2) \stackrel{?}{=} y_{2,1} \\ f(x_3) \stackrel{?}{=} y_{3,1} \end{array}$$

Figure 110: The Lamport scheme used to sign the message $m = 011$. Computing $f(x)$ is easy, but $f^{-1}(y)$ is hard.

24 Lecture25

Introduction to Quantum Information Processing

25 Lecture26

Quantum Key Distribution