

Introduction to Cryptography

Instructors: Xiang-Yang Li
Lan Zhang
Professor, CS Department

Introduction to Number Theory

➤ Divisors

- $b|a$ if $a=mb$ for an integer m
- $b|a$ and $c|b$ then $c|a$
- $b|g$ and $b|h$ then $b|(mg+nh)$ for any integers m,n

➤ Prime number

- An integer p is a prime number if it has only positive divisors 1 and p

➤ Relatively prime numbers p and q

- No common positive divisors for p and q except 1

Prime numbers

➤ Upto 200

- 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103
107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
199

➤ How many prime numbers are there?

- Infinity ---- Euclid gave simple proof
 - Proof by contradiction
 - They were also irregularly placed (arbitrary gap)
- How many in the range $[0, n]$ $\rightarrow \Theta(n / \log n)$
 - Approximately, the n th prime $\sim n \log n$
- How many primes with d bits approximately? $\sim \Theta(2^d / d)$

Extended Euclidean Algorithm

- input are two integers a and b , computes
 - their greatest common divisor (gcd) as well as
 - integers x and y such that $ax + by = \gcd(a, b)$.
- It later can also be used to compute the inverse of an integer

$$a^{-1} \bmod n$$

Modular Arithmetic

➤ Congruence

- $a \equiv b \pmod{n}$ says
 - when divided by n that a and b have the same remainder
- It defines a **relationship** between all integers
 - $a \equiv a$
 - $a \equiv b$ then $b \equiv a$ (symmetric)
 - $a \equiv b, b \equiv c$ then $a \equiv c$ (transitive)

Addition and Multiplication

- Integers modulo n with addition and multiplication form a commutative **ring** with the laws of
 - Associativity
 - $(a+b)+c \equiv a+(b+c) \pmod{n}$
 - Commutativity
 - $a+b \equiv b+a \pmod{n}$
 - Distributivity
 - $(a+b)*c \equiv (a*c)+(b*c) \pmod{n}$

4th arithmetic operation: Division

➤ Division

- $b/a \bmod n$
- multiplied by inverse of a : $b/a = b * a^{-1} \bmod n$
- $a^{-1} * a \equiv 1 \bmod n$
- $3^{-1} \equiv 7 \bmod 10$ because $3 * 7 \equiv 1 \bmod 10$
- What is the value of $20/3 \bmod 7$?
- Value of $1/3 \bmod 6$?

- Inverse does not always exist!
 - Only when $\gcd(a,n)=1$

Euclid's Extended GCD Routine

- Theorem:
 - If $\gcd(a,n)=1$ then the inverse of a mod n always exists
- Can extend Euclid's algorithm to find inverse by keeping track of $g_i = u_i \cdot n + v_i \cdot a$
- Extended Euclid's (or binary GCD) algorithm to find inverse of a number a mod n (where $(a,n)=1$) is:

When inverse exists

- If $\gcd(a,n)=1 \rightarrow$ inverse exists
 - We can find x, y such that $ax+ny=1$
 - Then $x = a^{-1} \bmod n$
- If inverse exists $\rightarrow \gcd(a,n)=1$
 - Let x be the inverse of a , i.e., $ax=1 \bmod n$
 - Then $xa=1+qn$ for some integer q
 - Let $\gcd(a,n)=d$. Then $d \mid (xa-qn)$
 - Obviously $d=1$ since $xa-qn=1$

Common arithmetic operations

➤ Exponentiation

- Given a , e , and p , find $b = a^e \bmod p$
- How to compute this efficiently?

➤ Discrete Logarithms

- Given a , b , and p , find x where $a^x = b \bmod p$
- Value of x is denoted as $(\log_a b \bmod p)$
- How to compute this efficiently?

Efficient computing of exponential

➤ Compute $a^b \bmod n$ efficiently when b, n large?

- Example: compute $a^{1024} \bmod 2^{1024} + 1$
- Simple approach: repetitively time a 1024 times?
- Efficient computation:
 - Write number b in binary format as $x_k x_{k-1} x_{k-2} \dots x_2 x_1 x_0$
 - Let $t_1 = a \bmod n$. Then compute $t_{i+1} = t_i * t_i \bmod n$ for $i < k$
 - Then

$$a^b \bmod n = a^{[x_k x_{k-1} x_{k-2} \dots x_2 x_1 x_0]} \bmod n$$

$$= \prod_{0 \leq i \leq k} [a^{(2^i)}]^{x_i} \bmod n$$

$$= \prod_{0 \leq i \leq k} t_i^{x_i} \bmod n$$

Time complexity?

Relative primes

- Two numbers a and n are relative primes if
 - $\gcd(a,n)=1$
- Consider all integers $0 < a < n$
 - How many are relative prime to n ?
 - Equivalently, how many a such that $a^{-1} \bmod n$ exists
- Typically
 - $Z_n = \{0, 1, 2, \dots, n-1\}$: all integers $0 \leq a < n$
 - $Z_n^* = \{a \mid 0 \leq a < n, \gcd(a,n)=1\}$
 - All integers in Z_n that are co-prime with n
 - Also called reduced residue set mod n

Group formed by relative numbers

- $Z_n = \{0, 1, 2, \dots, n-1\}$: all integers $0 \leq a < n$
- $(Z_n, +)$ forms a group
- Need to show a special element “0”, and inverse of an element a

- $Z_n^* = \{a \mid 0 \leq a < n, \gcd(a, n) = 1\}$
- $(Z_n^*, *)$
- Need to show a special element “0”, and inverse of an element a

Euler Totient Function

- If consider arithmetic modulo n , then a **reduced set of residues** is a subset of the complete set of residues modulo n which are relatively prime to n
 - eg for $n=10$,
 - the complete set of residues is $\{0,1,2,3,4,5,6,7,8,9\}$
 - the reduced set of residues is $\{1,3,7,9\}$
- The number of elements in the reduced set of residues is called the **Euler Totient function** $\phi(n)$

cont

➤ Compute $\phi(n)$

- If factoring of n is known
 - $\phi(n) = n \prod (1 - 1/p_i)$ where p_i is its prime factor
- Otherwise
 - It is expensive!
 - But not proved yet

➤ computing $\phi(n)$ when knowing fact $n = pq$ but not the number p and q

- Conjectured to be a hard question
- But not proved yet.
- Equivalent to find p and q

Euler's Theorem

➤ Theorem: Let $\gcd(a,n)=1$ then

○ $a^{\phi(n)} \bmod n = 1$

➤ Proof:

○ consider all reduced residues x_i in

■ $Z_n^* = \{x \mid 0 < x < n, \gcd(x,n)=1\}$

○ Then $ax_i, 1 \leq i \leq \phi(n)$ also form reduced residues set

○ Using $\prod ax_i = \prod x_i \bmod n$

■ Using Z_n^* and aZ_n^* are same sets!


○ We have $a^{\phi(n)} \prod x_i = \prod x_i \bmod n$

○ Thus, $a^{\phi(n)} = 1 \bmod n$

■ Using the fact that $\prod x_i$ has inverse

Fermat's Little Theorem

- Theorem: Let p be a prime and $\gcd(a,p)=1$ then
 - $a^{p-1} \bmod p = 1$
- Proof: similar to the proof of Euler's theorem
 - But consider all integers in Z_p
- Generally, for any prime number p
 - $a^p \bmod p = a$ (true for any number a)
- Generally, for any number $n=pq$
 - $a^{\phi(n)+1} \bmod n = a$ (true for any number a)
 - Need to prove for the case $\gcd(a,n)>1$



Do it yourself

Randomized Methods

➤ Las Vegas Method

- Always produces correct results
 - (may not give answer sometimes)
- Runs in **expected** polynomial time

➤ Monte Carlo Method

- Always runs in polynomial time
- May produce incorrect results with a bounded probability
- Yes-Biased Monte Carlo Method
 - Answer **yes** is always correct, but the answer **no** may be wrong
- No-biased Monte Carlo Method
 - Answer **no** is always correct, but the answer **yes** may be wrong

AKS method: Deterministic Poly-Time Method

- In 2002 Agrawal, Kayal and Saxena found a relatively simple deterministic algorithm which relies on no unproved assumptions.
 - Agrawal, Head of CSE, IITK
 - Kayal, Saxena (B.E. 2002), Indian Institute of Technology, Kanpur (IITK), PhD IITK
 - 2006 Gödel Prize (in TCS every year from 1993) and the 2006 Fulkerson Prize (in DMath every 3 year from 1979)
 - There has been a long list of research efforts devoted to find deterministic polynomial time methods for testing primes

Primitive Root

➤ Order of integer $\text{ord}_n(a)$

- The order of a modulo n is the smallest positive k such that $a^k \equiv 1 \pmod{n}$

➤ Primitive Root

- Integer a is a primitive root of n if the order of a modulo n is $\phi(n)$
- Not all integers have primitive root
 - Example $n=pq$ for primes p and q
- Prime p has $\phi(p-1)$ primitive roots

Computing Square root mod p

- Given number a , find number x , $x^2 = a \pmod p$
- when p is a prime number
 - If $p \equiv 3 \pmod 4$, then $x = a^{(p+1)/4} \pmod p$ is a solution.
 - If $p \equiv 5 \pmod 8$, $a^{(p-1)/4} = 1 \pmod p$ then $x = a^{(p+3)/8} \pmod p$
 - If $p \equiv 5 \pmod 8$, $a^{(p-1)/4} = -1 \pmod p$ then $x = 2a(4a)^{(p-5)/8} \pmod p$
 - If $p \equiv 1 \pmod 8$,

Generally difficult in this case, but has a probabilistic method
Using non-quadratic residue (Tonelli -Shanks method)

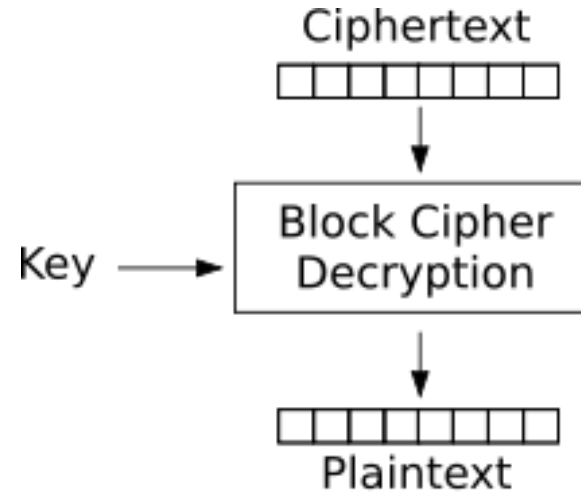
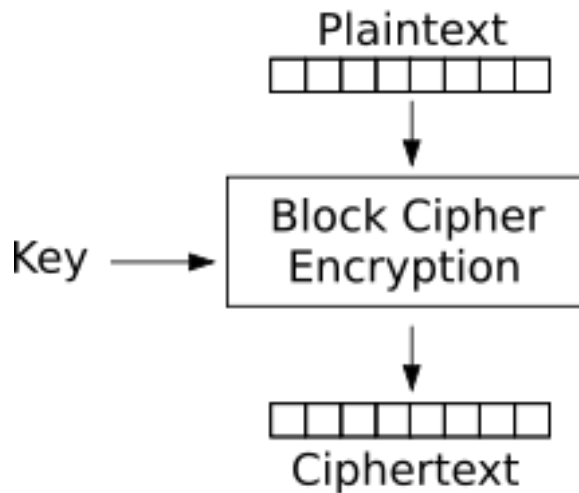
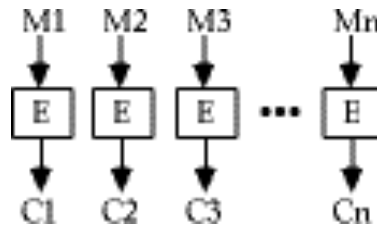
Some "hard" questions

- Some questions that are assumed to be hard, will be used as bases for cryptography
 - Integer factorization
 - Given n , find all its prime factors
 - Discrete logarithm
 - Given g , y , and p , find x such that $g^x \equiv y \pmod{p}$
 - Square root
 - Given b , find x such that $x^2 \equiv b \pmod{n}$. Here n is not a prime number, typically n is of format $p q$

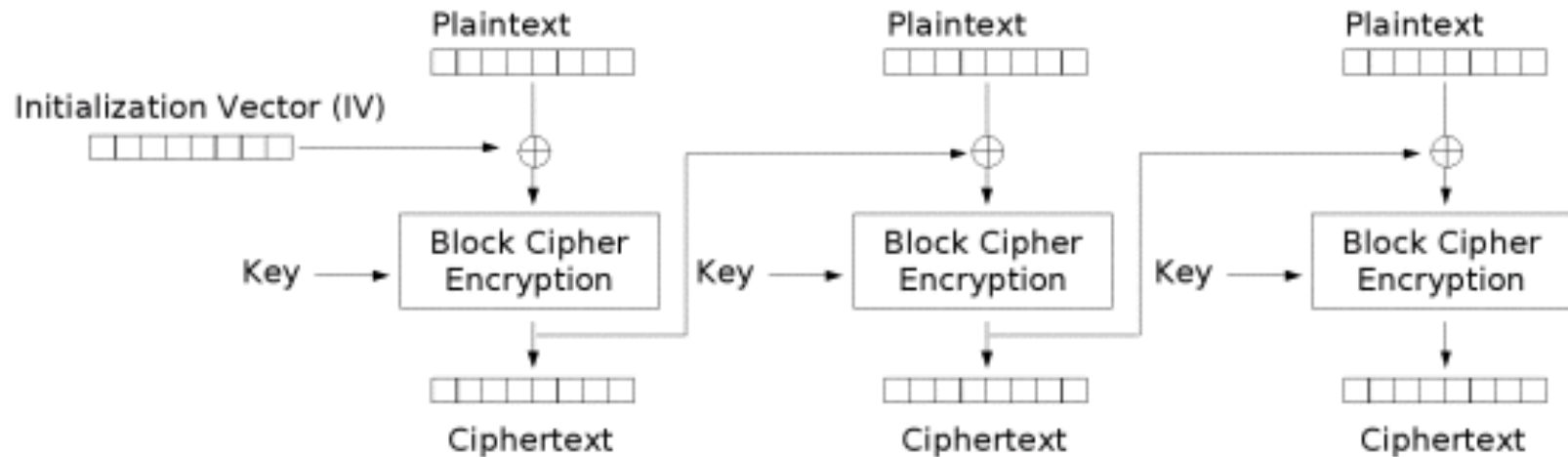
Block ciphers

Block Ciphers

- The message is broken into blocks,
 - Each of which is then encrypted
 - (Like a substitution on very big characters - 64-bits or more)



CBC Cipher (Cipher Block Chaining)

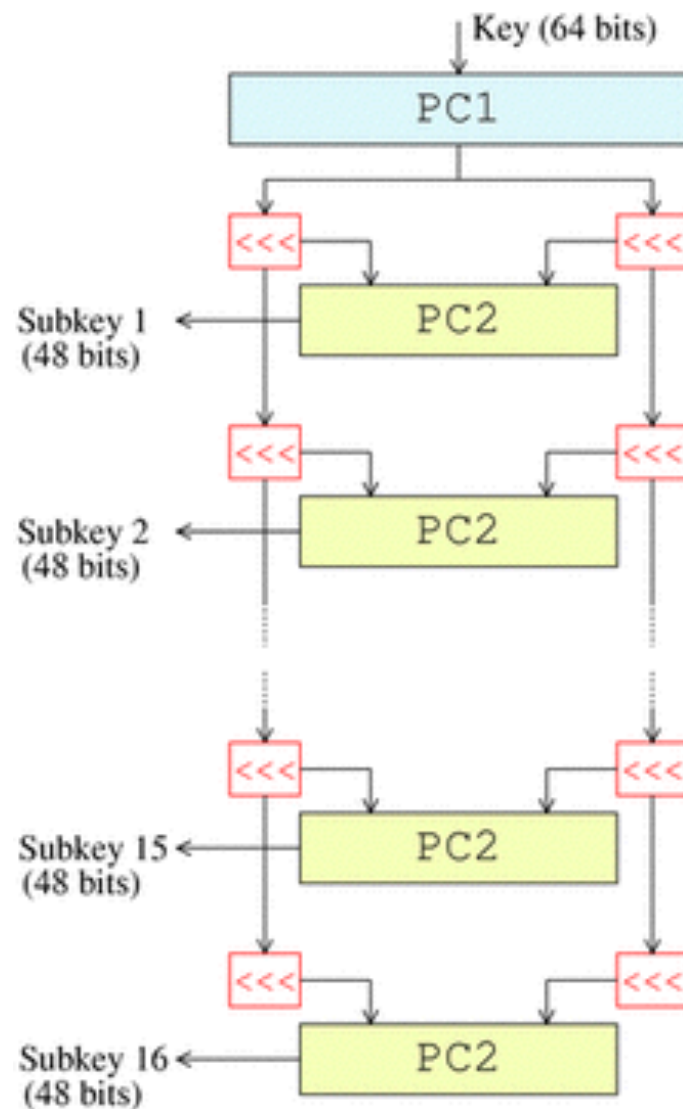
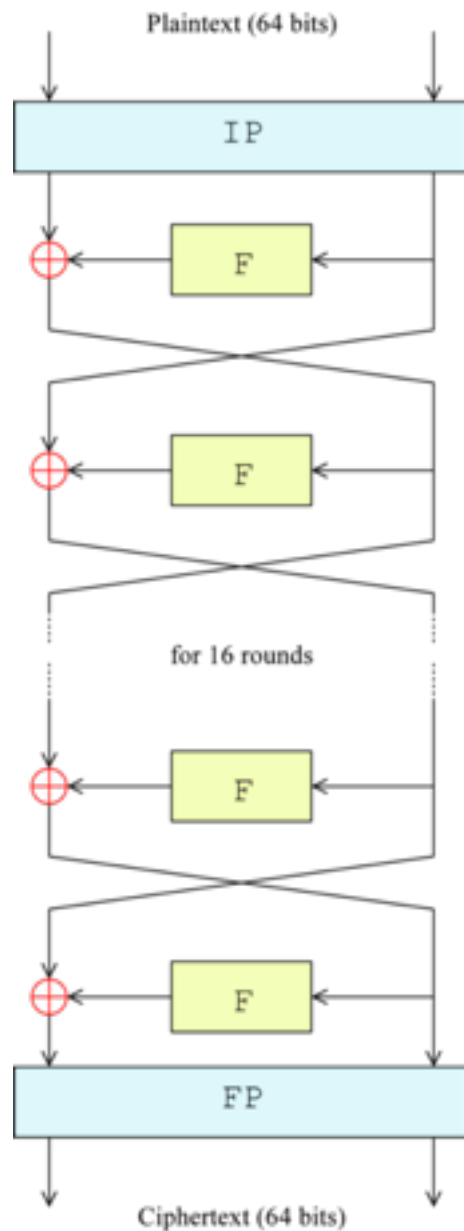


Cipher Block Chaining (CBC) mode encryption

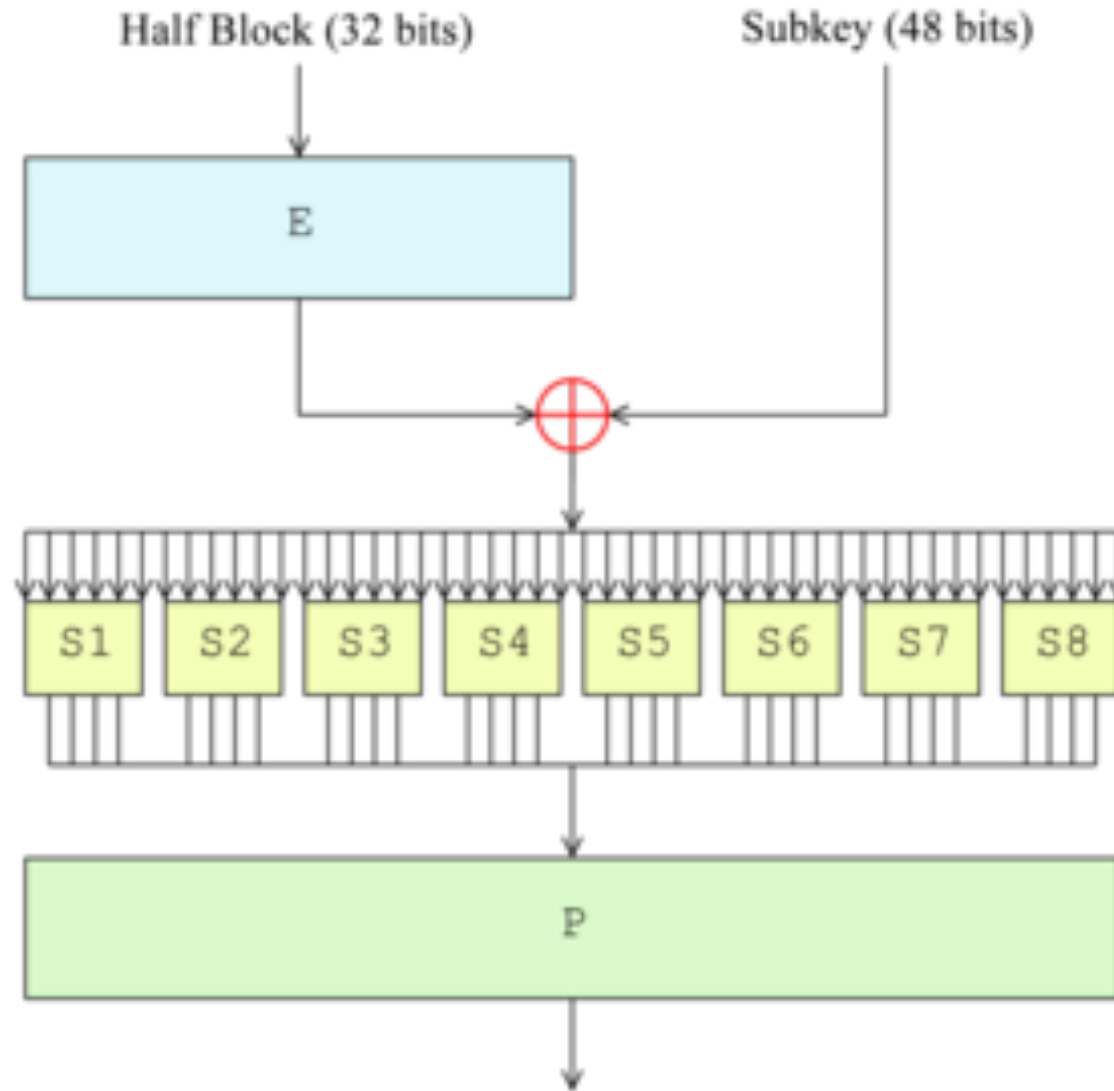
Data Encryption Standard

- Adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology
- Data are encrypted in 64-bit blocks using a 56-bit key
- The same algorithm is used for decryption.
- Subject to much controversy

DES



DES function f



Possible Techniques for Improving DES

- Multiple enciphering with DES
- Extending DES to 128-bit data paths and 112-bit keys
- Extending the key expansion calculation

Double DES?

- Using two encryption stages and two keys
 - $C = E_{k2}(E_{k1}(P))$
 - $P = D_{k1}(D_{k2}(C))$
- It is proved that there is no key $k3$ such that
 - $C = E_{k2}(E_{k1}(P)) = E_{k3}(P)$
- But Meet-in-the-middle attack
 - Thus 2DES is NOT secure (if DES is broken)

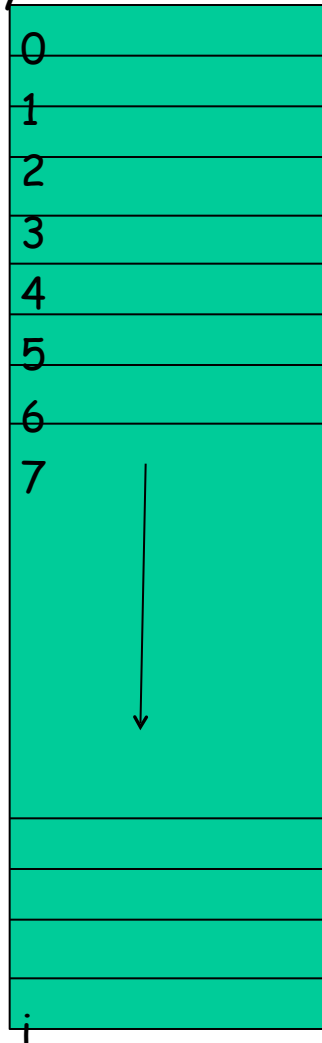
Meet-in-the-Middle Attack

- Assume $C = E_{k_2}(E_{k_1}(P))$
- Given the plaintext P and ciphertext C
- Encrypt P using all possible keys k_1
- Decrypt C using all possible keys k_2
 - Check the result with the encrypted plaintext lists
 - If found match, they test the found keys again for another plaintext and ciphertext pair
 - If it turns correct, then find the keys
 - Otherwise keep decrypting C

Breaking 2DES

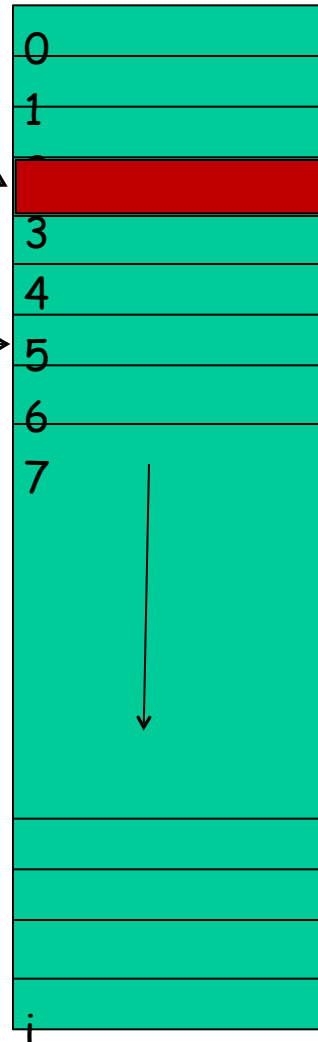
Decrypt Ciphertext C
using all keys

Encrypt messages M
keys



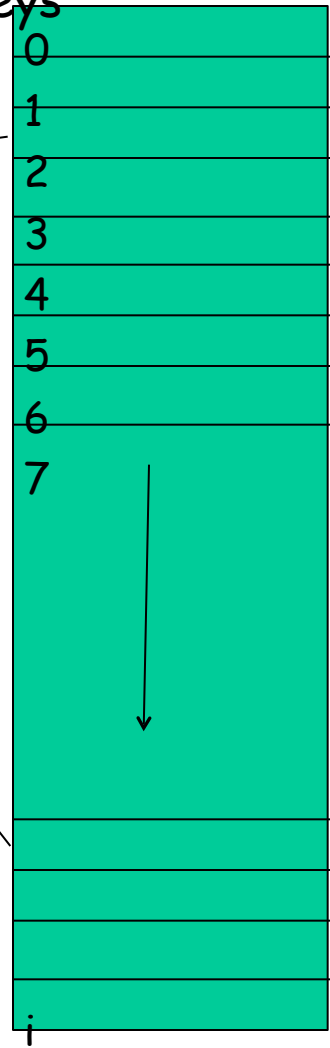
$i+1$

Middle results



$i+1$

keys



$i+1$

Advanced Encryption Standard



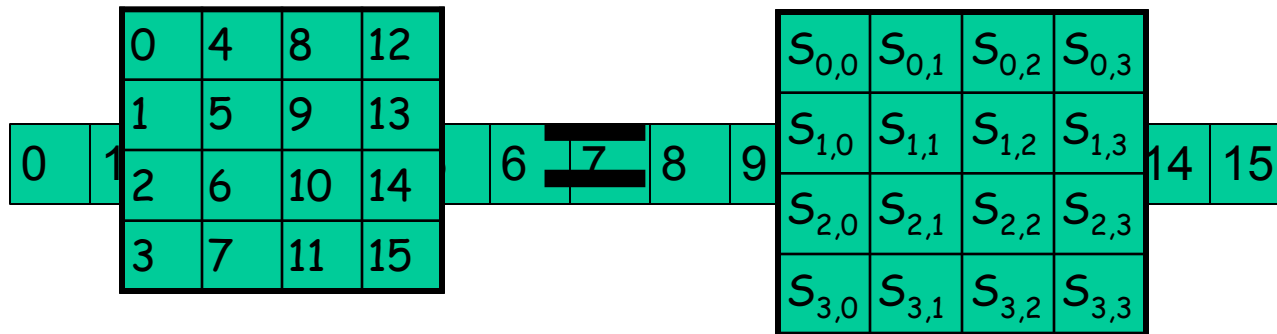
Not “American”
Encryption Standard

AES methods

- Convert to state array
- Transformations (and their inverses)
 - AddRoundKey
 - SubBytes
 - ShiftRows
 - MixColumns
- Key Expansion

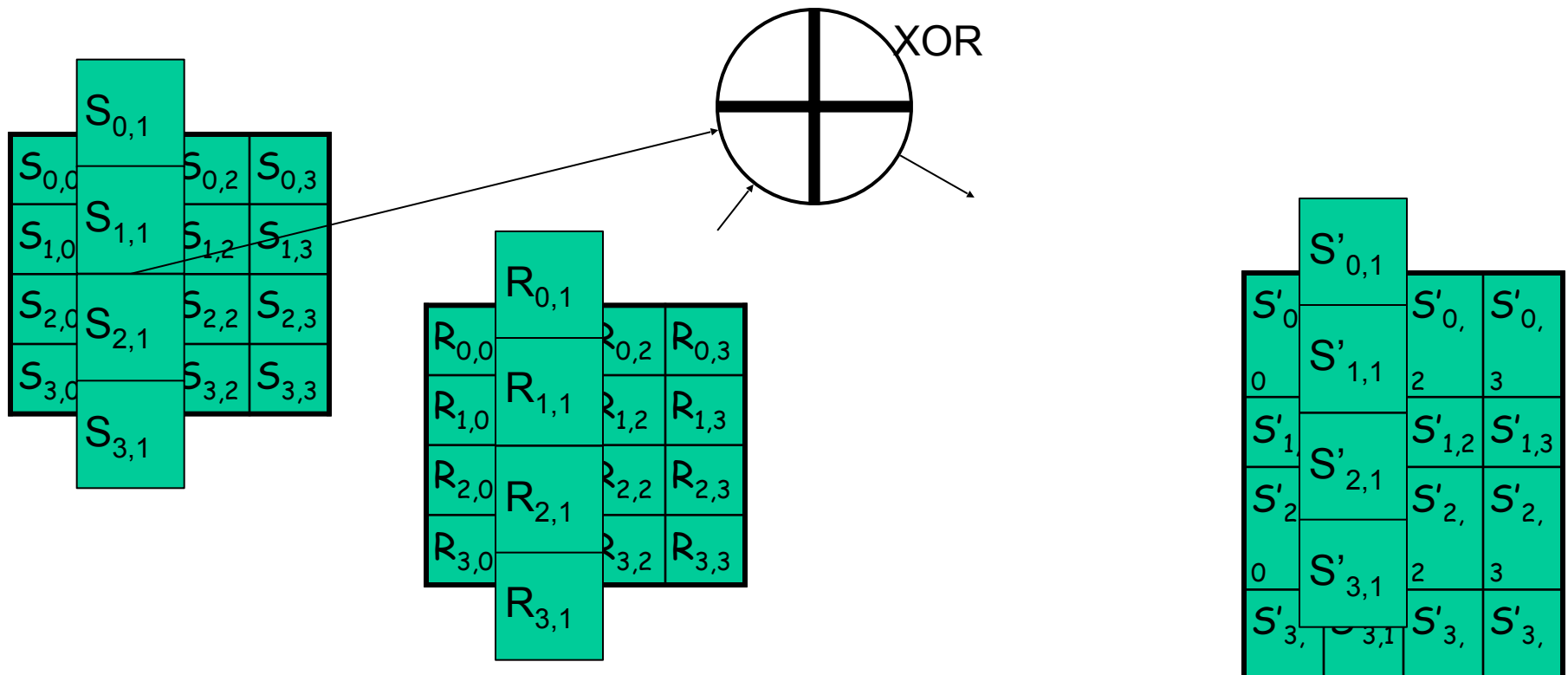
Convert to State Array

Input block:



AddRoundKey

- XOR each byte of the round key with its corresponding byte in the state array



SubBytes

- Replace each byte in the state array with its corresponding value from the S-Box

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

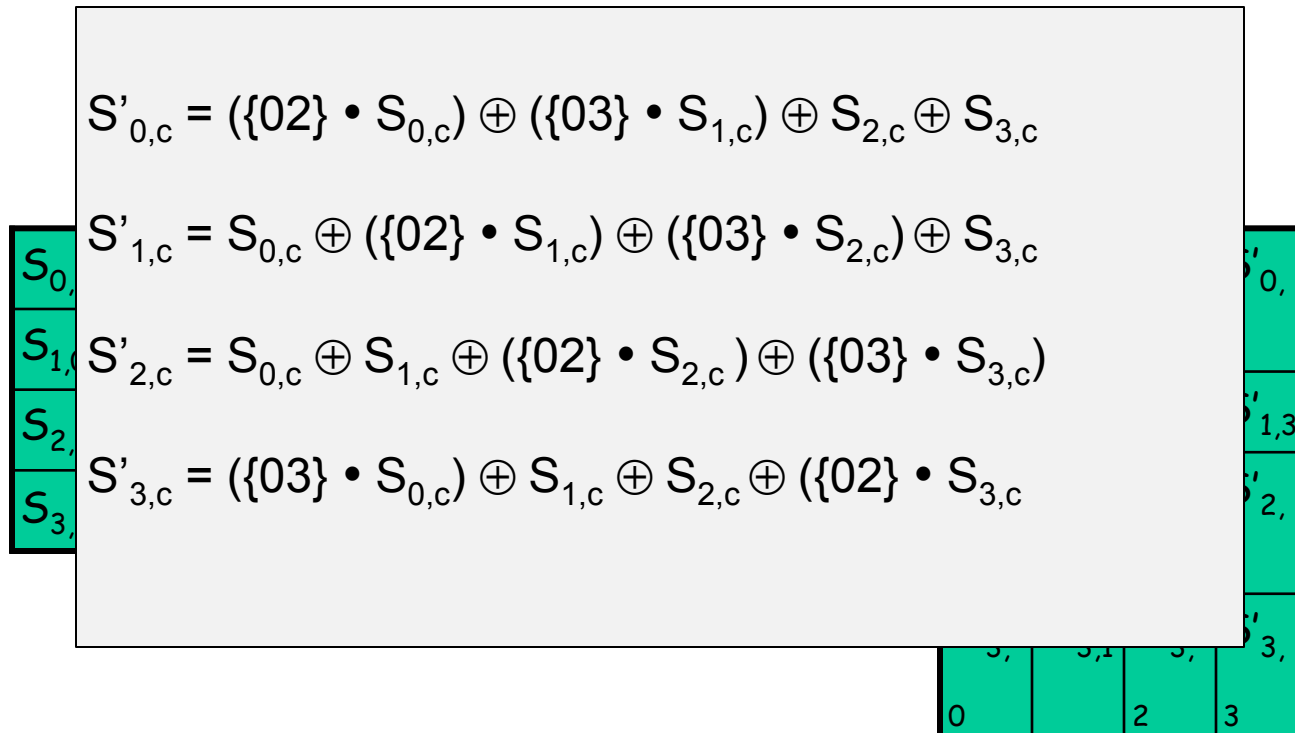
ShiftRows

- Last three rows are cyclically shifted

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

MixColumns

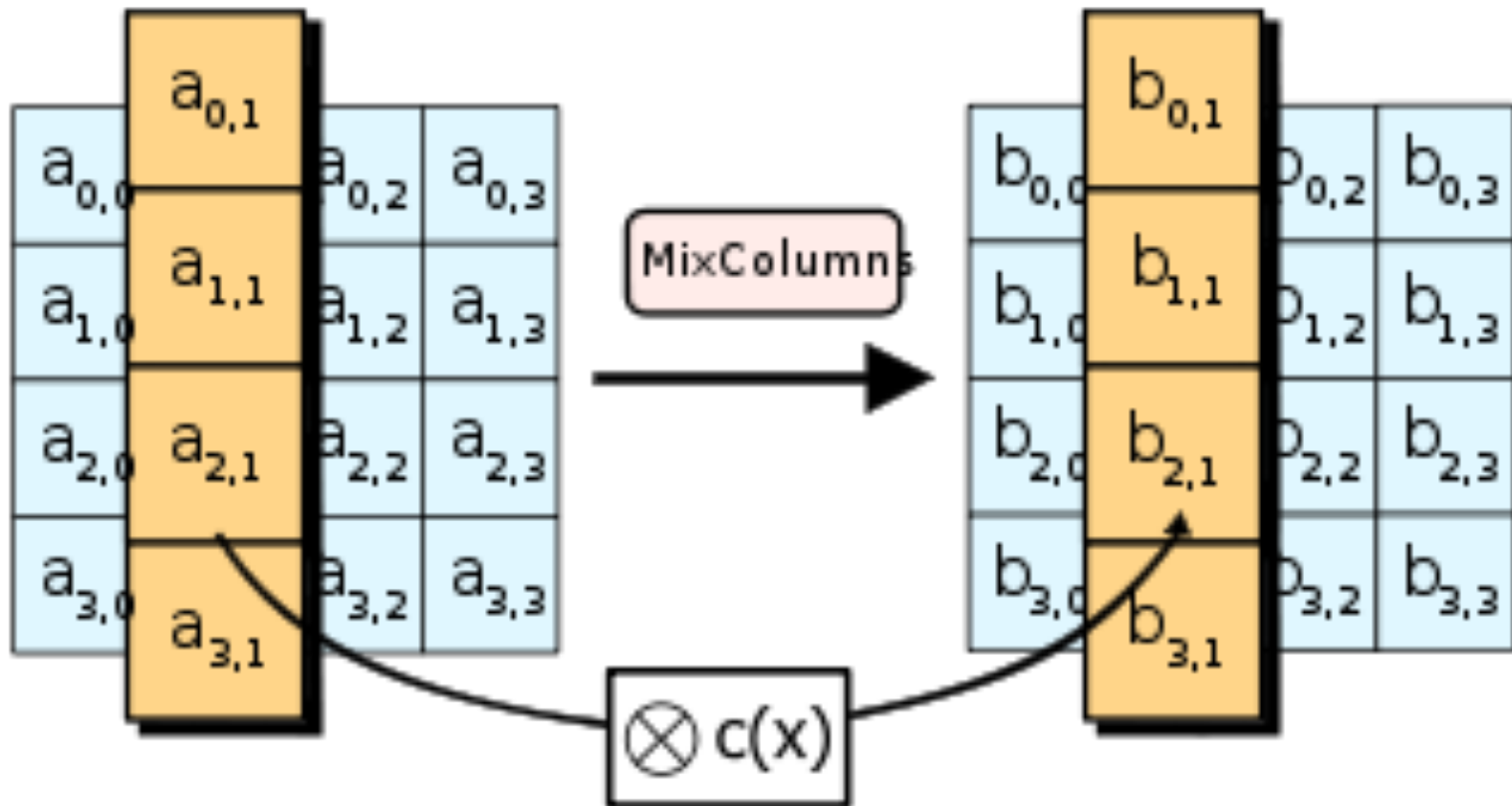
- Apply MixColumn transformation to each column



The diagram illustrates the MixColumns transformation. A central light gray box contains four equations for the transformed state elements $S'_{i,c}$ based on the original state elements $S_{i,c}$ and fixed constants {02} and {03}. To the left and right of this box are vertical columns of teal boxes representing the state elements $S_{0,c}$ through $S_{3,c}$ and their transformed counterparts $S'_{0,c}$ through $S'_{3,c}$. At the bottom right, a horizontal row of teal boxes shows the column indices 0, 1, 2, and 3.

$$\begin{aligned} S'_{0,c} &= (\{02\} \cdot S_{0,c}) \oplus (\{03\} \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S'_{1,c} &= S_{0,c} \oplus (\{02\} \cdot S_{1,c}) \oplus (\{03\} \cdot S_{2,c}) \oplus S_{3,c} \\ S'_{2,c} &= S_{0,c} \oplus S_{1,c} \oplus (\{02\} \cdot S_{2,c}) \oplus (\{03\} \cdot S_{3,c}) \\ S'_{3,c} &= (\{03\} \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \cdot S_{3,c}) \end{aligned}$$

Matrix multiplication view



Public key system

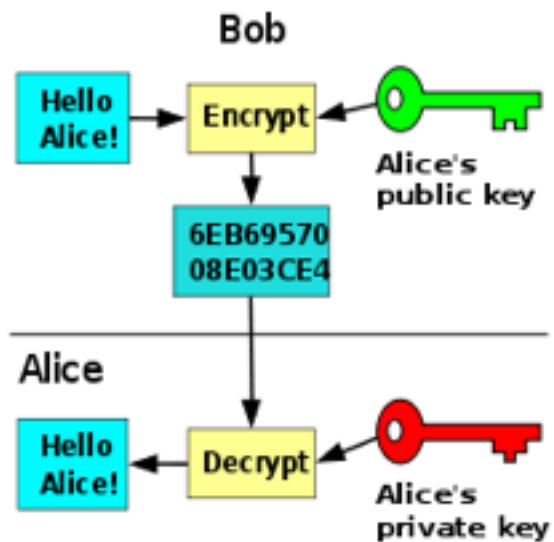
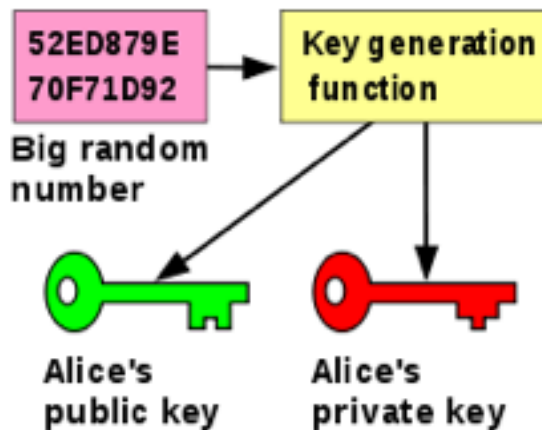
Public Key Encryption

- Two difficult problems
 - Key distribution under conventional encryption
 - Digital signature
- Diffie and Hellman, 1976
 - Astonishing breakthrough
 - One key for encryption and the other related key for decryption
 - It is computationally infeasible (under some assumptions) to determine the decryption key using only the encryption key and the algorithm

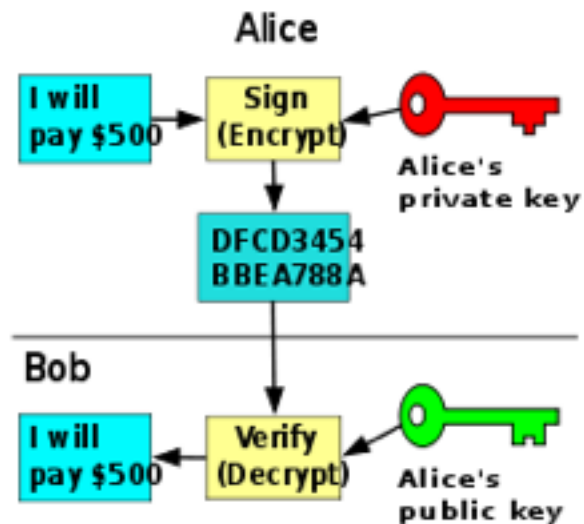
Public Key Cryptosystem

- Essential steps of public key cryptosystem
 - Each end generates a pair of keys
 - One for encryption and one for decryption
 - Each system publishes one key, called public key, and the companion key is kept secret
 - If A wants to send message to B
 - Encrypt it using B's public key
 - When B receives the encrypted message
 - It decrypt it using its own private key

Alice

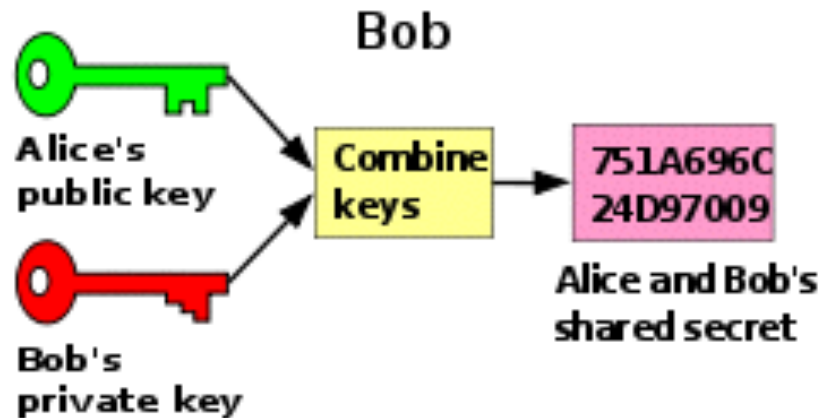
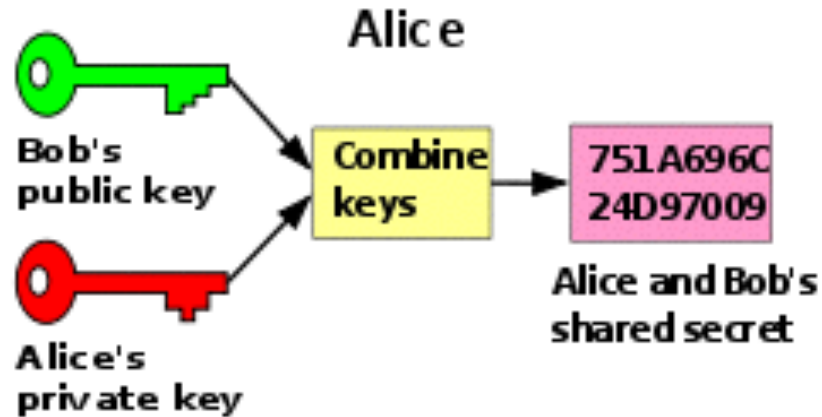


Encryption



Digital signature

Key distribution



Applications of PKC

➤ Encryption/Decryption

- The sender encrypts the message using the receiver's public key
 - Q: Why not use the sender's secret key?

➤ Digital signature

- The sender signs a message by encrypt the message or a transformation of the message using its own private key

➤ Key exchange

- Two sides cooperate to exchange a session key, typically for conventional encryption

Conditions of PKC

➤ **Computationally easy**

- To generate public and private key pair
- To encrypt the message using encryption key
- To decrypt the message using decryption key

➤ **Computational infeasible**

- To compute the private key using public key
- To recover the plaintext using ciphertext and public key

➤ **The encryption and decryption can be applied in either order**

RSA Algorithm

- R. Rivest, A. Shamir, L. Adleman (1977)
 - James Ellis came up with the idea in 1970, and proved that it was theoretically possible. In 1973, Clifford Cocks a British mathematician invented a variant on RSA; a few months later, Malcom Williamson invented a Diffie-Hellman analog
 - Only revealed till 1997
- Patent expired on September 20, 2000.
- Block cipher using integers $0 \sim n-1$
 - Thus block size k is less than $\log_2 n$
- Algorithm:
 - Encryption: $C = M^e \bmod n$
 - Decryption: $M = C^d \bmod n$
- Both sender and the receiver know n

RSA (public key encryption)

- Alice wants Bob to send her a message. She:
 - selects two (large) primes p, q , **TOP SECRET**,
 - computes $n = pq$ and $\phi(n) = (p-1)(q-1)$,
 $\phi(n)$ also **TOP SECRET**,
 - selects an integer e , $1 < e < \phi(n)$, such that
 $\gcd(e, \phi(n)) = 1$,
 - computes d , such that $de \equiv 1 \pmod{\phi(n)}$,
 d also **TOP SECRET**,
 - gives **public key** (e, n) , keeps **private key** (d, n) .

Requirements

- Possible to find e and d such that
 - $M = M^{de} \bmod n$ for all message M
- Easy to conduct encryption and decryption
- Infeasible to compute d
 - Given n and e

Security of RSA

- Brute force: try all possible private keys
- Factoring integer n , then know $\phi(n)$
 - Not proven to be NPC
- Determine $\phi(n)$ directly without factoring
 - Equivalent to factoring! (1996)
 - Two variables and two equations
$$\begin{cases} \phi(n) = (p-1)(q-1) \\ n = pq \end{cases}$$
- Determine d directly without knowing $\phi(n)$
 - Currently appears as hard as factoring
 - But not proven, so it may be easier!

Other attacks on RSA

➤ Comprised decryption key

- If the private key d (for decryption of received ciphertext) of a user is comprised, then the user has to reselect n and e and d
- It cannot use the old number n to produce the key-pairs!
- Otherwise attacker already can factor n almost surely!

➤ The number n can only be used by one person

- If two user uses the same n , even they do not know the factoring of n , they still could figure out the factoring of n with probability almost one.
 - Similar as above

Factoring algorithm given the decryption key d

Algorithm *Factor*(n, e, d) // given n , e , and d , find a factor of n

Choose a random w in $[1, n - 1]$

compute $x = \gcd(w, n)$

if $1 < x < n$ then output x as a factor, and quit

write $ed - 1 = 2^s r$, where r is odd

compute $v = w^r \bmod n$

If $v = 1 \bmod n$, then quit and failed

while $v \neq 1 \bmod n$ do

$v_0 = v$;

$v = v^2 \bmod n$

If $v_0 = -1 \bmod n$ then quit and failed

else compute $x = \gcd(v_0 + 1, n)$ as one factor of n

Bit security of RSA

- Given ciphertext C ,
 - We may want to find the last bit of M , denoted by $\text{parity}(C)$
 - We may want to find if $M > n/2$, denoted by $\text{half}(C)$
 - We may want to find all bits of M
- The above three attacks are the same!
 - If we can solve one, we can solve the other two!

Bit security of RSA

$$C = M^e \bmod n$$

Then the encryption of $2M$ is $(2M)^e \bmod n = 2^e M^e \bmod n = 2^e C \bmod n$

If $M < n/2$, then $2M < n$ and $(2M \bmod n)$ is an even number

If $M > n/2$, then $2M > n$ and $(2M \bmod n = 2M - n)$ is an odd number

Thus, when $M < n/2$, the last bit of the message returned by $\text{parity}(2^e C)$ will be 0;

when $M > n/2$, the last bit of the message returned by $\text{parity}(2^e C)$ will be 1.

1. The following two are equivalent

1. Find the last bit of M , denoted by $\text{parity}(C)$
2. Find if $M > n/2$, denoted by $\text{half}(C)$

Thus, if we can solve $\text{parity}(C)$, then we can use $\text{parity}(2^e C)$ to answer whether $M > n/2$ or not.

Then, consider the other direction. Assume that we can solve $\text{half}(C)$.

Then we use the following property

1) If M is an even number, then it is easy to show that $\frac{M}{2} = M * 2^{-1} \bmod n$

In other words, $M * 2^{-1} \bmod n$ is an integer $< n/2$ when M is even.

2) When M is an odd number, assume that $M * 2^{-1} \equiv x \bmod n$.

Then we have $2 * M * 2^{-1} \equiv 2x \bmod n, \Rightarrow M \equiv 2x \bmod n$

Since M is an odd number, obviously, $x > n/2$.

Thus, $M * 2^{-1} \bmod n$ is an integer $> n/2$ when M is odd

Notice that the encryption of $M * 2^{-1} \bmod n$ is $C * (2^{-1} \bmod n) \bmod n$

Thus, $\text{half}(C * (2^{-1} \bmod n) \bmod n)$ returns "message is $> n/2$ " $\Rightarrow M$ is odd

$\text{half}(C * (2^{-1} \bmod n) \bmod n)$ returns "message is $< n/2$ " $\Rightarrow M$ is even

Bit security of RSA

1. The following two are equivalent
 1. Find the last bit of M , denoted by $\text{parity}(C)$
 2. Find all bits of M

$$C = M^e \bmod n$$

Then the encryption of $2M$ is $(2M)^e \bmod n = 2^e M^e \bmod n = 2^e C \bmod n$

If $M < n/2$, then $2M < n$ and $(2M \bmod n)$ is an even number

If $M > n/2$, then $2M > n$ and $(2M \bmod n = 2M - n)$ is an odd number

Thus, when $M < n/2$, the last bit of the message returned by $\text{parity}(2^e C)$ will be 0;

when $M > n/2$, the last bit of the message returned by $\text{parity}(2^e C)$ will be 1.

Thus, if we can solve $\text{parity}(C)$, then we can use $\text{parity}(2^e C)$ to answer whether $M > n/2$ or not.

We continue this by checking $\text{parity}(2^{2e} C) \Rightarrow$ tells whether $2M \bmod n > n/2$ or not

We continue this by checking $\text{parity}(2^{ie} C) \Rightarrow$ tells whether $2^{i-1} M \bmod n > n/2$ or not

Other Public Key Systems

➤ Rabin Cryptosystem

- Decryption is not unique

➤ Elgamal Cryptosystem

- Expansion of the plaintext (double)

➤ Elliptic Curve System

- If directly implement Elgamal on elliptic curve
 - Expansion of plaintext by 4; Restricted plaintext
- Menezes-Vanston system is more efficient

ElGamal Cryptosystem

➤ Based on Discrete Logarithm

- Find unique integer x such that $g^x = y \bmod p$
 - Here g is a primitive element in Z_p , p is prime

➤ Procedure

- Make p, g, y public, keep x secret
- Encryption:
 - $E_k(m) = (g^k \bmod p, m \cdot y^k \bmod p)$
- Decryption
 - $D_k(y_1, y_2) = y_2 (y_1^x)^{-1} \bmod p$

Security of ElGamal

- ElGamal is a simple example of a **semantically secure** asymmetric key encryption algorithm (under reasonable assumptions).
- ElGamal's security rests, in part, on the difficulty of solving the discrete logarithm problem in G .
 - Specifically, if the discrete logarithm problem could be solved efficiently, then ElGamal would be broken. However, the security of ElGamal actually relies on the so-called Decisional Diffie-Hellman (DDH) assumption. This assumption is often stronger than the discrete log assumption, but is still believed to be true for many classes of groups.

Homomorphic

- combine several ciphertexts together in a useful way to produce a related ciphertext.
- In ElGamal and in RSA, one can combine encryptions of m_1 and m_2 to obtain a valid encryption of their product m_1m_2 .

FHE

- Each of the examples listed above allows homomorphic computation of only one operation (either addition or multiplication) on plaintexts.
- A cryptosystem which supports both addition and multiplication (thereby preserving the ring structure of the plaintexts) is known as fully homomorphic encryption (FHE) and is far more powerful
 - fully homomorphic cryptosystem would have great practical implications in the outsourcing of private computations, for instance, in the context of cloud computing.
 - Craig Gentry using lattice-based cryptography showed the first fully homomorphic encryption scheme as announced by IBM on June 25, 2009
 - http://en.wikipedia.org/wiki/Homomorphic_encryption#Fully_homomorphic_encryption

Digital Signature

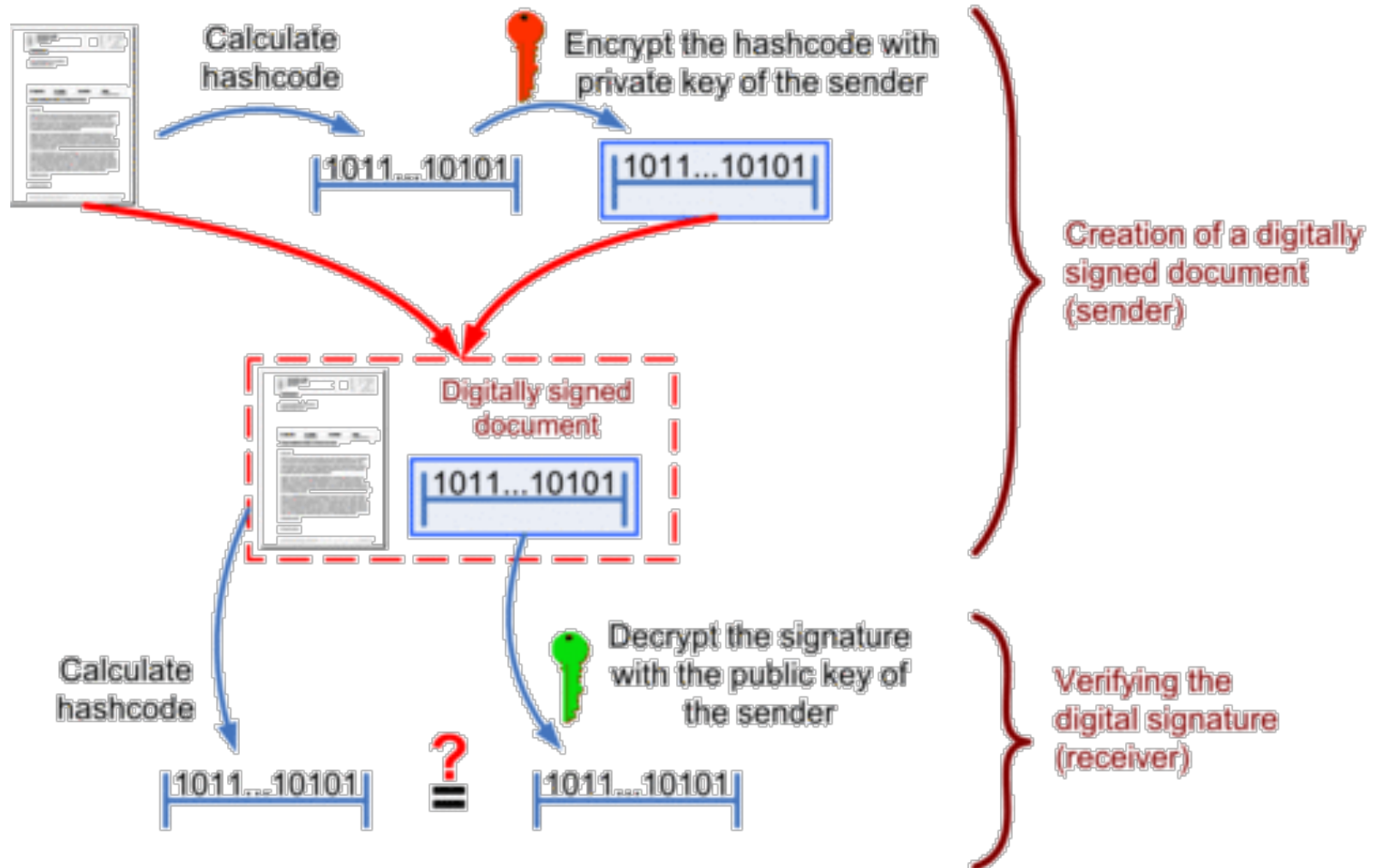
Hash Function

- Map a message to a smaller value
- Requirements
 - Be applied to a block of data of any size
 - Produced a fixed length output
 - $H(x)$ is easy to compute (by hardware, software)
 - **One-way**: given code h , it is computationally infeasible to find x :
 $H(x)=h$
 - Preimage resistance
 - **Weak collision resistance**: given x , computationally infeasible to find y so $H(x)=H(y)$
 - Second preimage resistance
 - **Strong collision resistance**: Computationally infeasible to find x, y so $H(x)=H(y)$
 - Collision resistance

Hash Algorithms

- See similarities in the evolution of hash functions & block ciphers
 - increasing power of brute-force attacks
 - leading to evolution in algorithms
 - from DES to AES in block ciphers
 - from MD4 & MD5 to SHA-1 SHA-2, SHA-3 algorithms
- likewise tend to use common iterative structure as do block ciphers

Creating and verifying a digital signature



If the calculated hashcode does not match the result of the decrypted signature, either the document was changed after signing, or the signature was not generated with the private key of the alleged sender.

Securities

- A **total break** results in the recovery of the signing key.
- A **universal forgery** attack results in the ability to forge signatures for any message.
- A **selective forgery** attack results in a signature on a message of the adversary's choice.
- An **existential forgery** merely results in some valid message/signature pair not already known to the adversary.

ElGamal Signature

- Global public components
 - Prime number p with 512-1024 bits
 - Primitive element g in Z_p
- Users private key
 - Random integer x less than p
- Users public key
 - Integer $y = g^x \bmod p$

Elgamal

➤ Signature

- For each message M , generates random k
- Computes $r = g^k \bmod p$
- Computes $s = k^{-1}(H(M) - xr) \bmod (p-1)$
- Signature is (r, s)

➤ Verifying

- Computes $v_1 = g^{H(M)} \bmod p$
- Computes $v_2 = y^r r^s \bmod p$
- Test if $v_1 = v_2$

Digital Signature Standard

- FIPS PUB 186 by NIST, 1991
- Final announcement 1994
- It uses
 - Secure Hashing Algorithm (SHA) for hashing
 - Digital Signature Algorithm (DSA) for signature
 - The hash code is set as input of DSA
 - The signature consists of two numbers
- DSA
 - Based on the difficulty of discrete logarithm
 - Based on Elgamal and Schnorr system

DSA

- Global public components
 - Prime number p with 512-1024 bits
 - Prime divisor q of $(p-1)$ with 160 bits
 - Integer $g = h^{(p-1)/q} \bmod p$
- Users private key
 - Random integer x less than q
- Users public key
 - Integer $y = g^x \bmod p$

DSA

➤ Signature

- For each message M , generates random k
- Computes $r = (g^k \bmod p) \bmod q$
- Computes $s = k^{-1}(H(M) + xr) \bmod q$
- Signature is (r, s)

➤ Verifying

- Computes $w = s^{-1} \bmod q$, $u_1 = H(M)w \bmod q$
- Computes $u_2 = rw \bmod q$, $v = (g^{u_1}y^{u_2} \bmod p) \bmod q$
- Test if $v = r$

Blind Signature (digital cash)

- first introduced by Chaum, allow a person to get a message signed by another party without revealing any information about the message to the other party.

Blind Signature (digital cash)

- Suppose Alice has a message m that she wishes to have signed by Bob, and she does not want Bob to learn anything about m .
 - Let (n, e) be Bob's public key and (n, d) be his private key.
 - Alice generates a random value r such that $\gcd(r, n) = 1$ and sends $x = (r^e m) \bmod n$ to Bob. The value x is "blinded" by the random value r ; hence Bob can derive no useful information from it.
 - Bob returns the signed value $t = x^d \bmod n$ to Alice.
 - Since $x^d \equiv (r^e m)^d \equiv r m^d \bmod n$,
 - Alice can obtain the true signature s of m by computing
$$s = r^{-1} t \bmod n.$$

Secret Sharing

Threshold Scheme

- A (t,w) -threshold scheme
 - Sharing key K among a set of w users
 - Any t users can recover the key
 - Any $t-1$ users can not do so
- Schemes
 - Shamir's scheme
 - Geometric techniques
 - Matroid theory

Shamir's Scheme

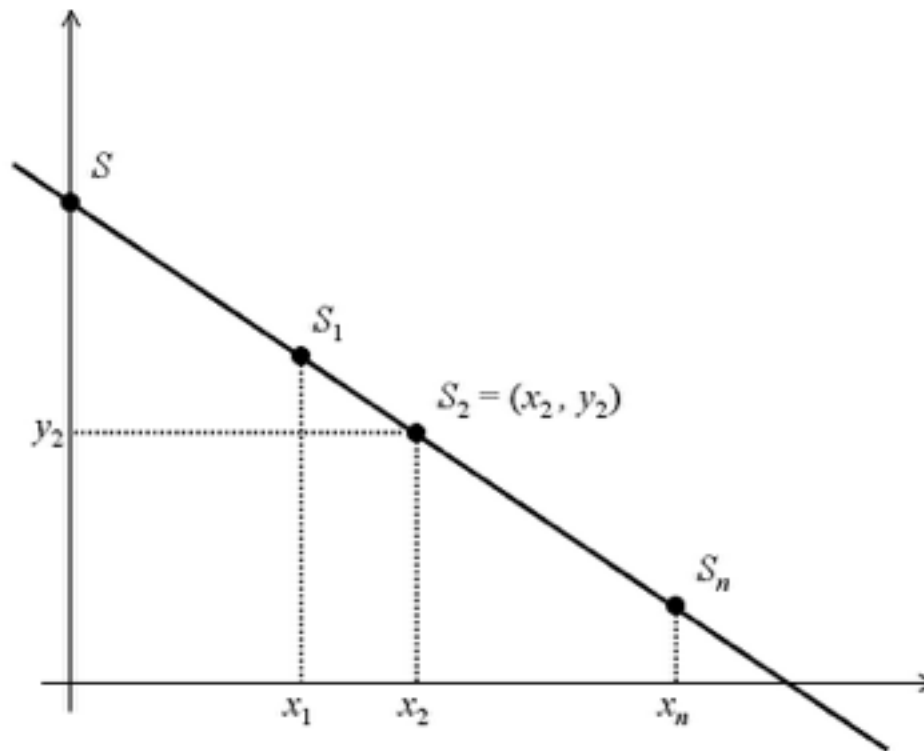
➤ Initialization phase

- Dealer chooses a large prime number p
- Dealer chooses w distinct x_i from Z_p
- Gives value x_i to person p_i

➤ Share distribution of key k from Z_p

- Dealer choose $t-1$ random number a_i
- Dealer computes $y_i = f(x_i)$
 - Here $f(x) = k + \sum a_j x^j \mod p$
- Dealer gives share y_i to person p_i

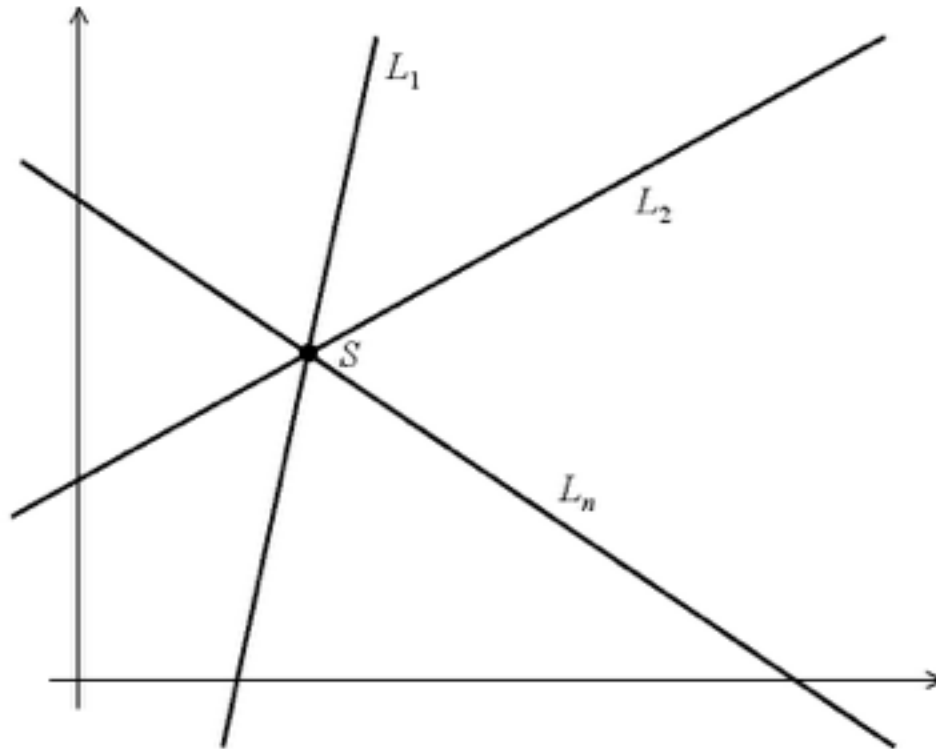
Geometry View



Blakley's Scheme

- Secret is a point in an t -dimensional space
- Dealer gives each user a hyper-plane passing the secret point
- Any t users can recover the common point

Geometry View



Verifiable Secret Sharing

- How can I know whether a share is correct or not?
 - Note that the correctness of a share can be verified using t other shares.
 - However, we can't ask other parties to reveal t shares.
- So each share should have a commitment which is public.
 - The correctness of shares can be verified using commitments.
 - This is called Verifiable Secret Sharing (VSS).

Interactive proof

Interactive Proof

- Interactive proof is a protocol between two parties in which one party, called the **prover**, tries to prove a certain fact to the other party, called the **verifier**
 - such as **the membership of a string x in a given language**, often with the goal of revealing no further details about this knowledge.
- The prover(s) and verifier are formally defined as *probabilistic Turing machines* with special "interaction tapes" for exchanging messages.
- Often takes the form of a challenge-response protocol

Desired Properties

➤ Desired properties of interactive proofs

- **Completeness**: The verifier always accepts the proof if the prover knows the fact and both the prover and the verifier follow the protocol.
- **Soundness**: Verifier always rejects the proof if prover does not know the fact, and verifier follows protocol.
- **Zero knowledge**: The verifier learns nothing about the fact being proved (except that it is correct) from the prover that he could not already learn without the prover. In a zero-knowledge proof, the verifier cannot even later prove the fact to anyone else.

Cont

➤ ZKP:

- if the statement is true, no cheating verifier learns anything other than this fact.
- This is formalized by showing that every cheating verifier has some *simulator* that, given only the statement to be proven (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.

Discrete Logarithm

➤ Question:

- Prover wants to prove to verifier that he knows x such that $y = g^x \bmod p$.
- Here g , y , and p are public information
- Prover does not want to publicize the value of x .

Proof Protocol

➤ Repeat the following for $\log_2 n$ times

- Prover

- Chooses random $j < p-1$ and computes $r = g^j \bmod p$. Sends r to verifier

- Verifier

- Chooses a random i from $\{0,1\}$, sends it to prover

- Prover

- Computes $h = i \times x + j \bmod p-1$, sends h to verifier

- Verifier

- Checks if $g^h = y^i r \bmod n$

- Accepts the proof if equation holds all $\log_2 n$ rounds

Cont

➤ **Correctness**

- Show that verifier will accept the prover if indeed knows

➤ **Soundness**

- Show that verifier will detect the prover if it does not know with a good probability

➤ **Zero-knowledge**

- Show that verifier gets nothing from the protocol

Bit Commitments

➤ Bit commitment

- Sometimes, it is desirable to give someone a piece of information, but not commit to it until a later date. It may be desirable for the piece of information to be held secret for a certain period of time.
- Example: stock up and down

Properties

➤ Bit commitment scheme

- The sender encrypts the b in some way
- The encrypted form of b is called blob
- Scheme $f: (X,b) \rightarrow Y$

➤ Properties

- Concealing: verifier cannot detect b from $f(x,b)$
- Binding: sender can open the blob by revealing x
- Hence, the sender must use random x to mask b

Methods

- One can choose any encryption method E
 - Function $f((x_0, k), b) = E_k((x_0, b))$
 - Need supply decryption k to reveal b
 - Assume the decryption method D is known
- Choose any integer $n=pq$, p and q are large primes
 - Function $f(x, b) = m^b x^2 \bmod n$
 - Goldwasser-Micali Scheme
 - Here $n=pq$, m is not quadratic residue, m, n public
 - $m x_1^2 \bmod n \neq x_2^2 \bmod n$
 - So sender can not change mind after commitment

Oblivious Transfer

➤ What is oblivious transfer

- Alice wants to send Bob a secret in such a way that Bob will know whether he gets it, but Alice won't.
- Another version is where Alice has several secrets and transfers one of them to Bob in such a way that Bob knows what he got, but Alice doesn't.
- This kind of transfer is said to be oblivious (to Alice).

Oblivious Transfer

- A simple cryptographic primitive first studied by Rabin.
 - Kilian showed that you can essentially base ANY cryptographic protocol on this primitive.
- Suppose Alice sends a message to Bob.
 - We want that Bob receives the message with probability $\frac{1}{2}$.
 - We also want that Alice does not know whether Bob receives it or not.

Transfer Factoring

- By means of RSA, oblivious transfer of any secret amounts to oblivious transfer of the factorization of $n=pq$
 - Bob chooses x and sends $x^2 \bmod n$ to Alice
 - Alice (who knows p, q) computes the square roots $x, -x, y, -y$ of $x^2 \bmod n$ and sends one of them to Bob. Note that Alice does not know x .
 - If Bob gets one of y or $-y$, he can factor n . This means that with probability $1/2$, Bob gets the secret. Alice doesn't know whether Bob got one of y or $-y$ because she doesn't know x .

Factoring

- If one knows x and y such that
 - 1) $x^2 = y^2 \pmod n$
 - 2) $0 < x, y < n$, $x \neq y$ and $x + y \not\equiv 0 \pmod n$
 - Number n is the product of two primes
- Then n can be factored
 - First $\gcd(x+y, n)$ is a factor of n
 - And $\gcd(x-y, n)$ is a factor of n

1-out-2 OT

- There are many variants of OT; 1-out-of-2 OT is a popular one.
 - Alice has two messages m_0 and m_1 .
 - Bob has a bit b (i.e., chooses to receive m_b).
 - Alice should not learn b .
 - Bob should not learn m_{1-b} .

Bellare-Micali Protocol (1)

- Let G be a cyclic group in which discrete log is hard; let g be a generator.
- Alice (or a public procedure) chooses y in G .
- Bob chooses x_b and computes $y_b = g^{x_b}$.
- Bob also computes $y_{1-b} = y / y_b$.

Bellare-Micali Protocol (2)

- Bob sends y_0, y_1 to Alice.
- Alice checks $y_0 y_1 = y$ and encrypts m_0, m_1 using ElGamal public parameters y_0, y_1 , respectively.
- Bob decrypts the encryption of m_b using x_b .

Simple 1-n OT

Algorithm 1: 1-out-of- z Oblivious Transfer (OT_z^1)

Initialization:

System parameters: (g, h, G_q) ;

Sender's input: $s_1, s_2, \dots, s_z \in G_q$;

Receiver's choice: $\alpha, 1 \leq \alpha \leq z$;

- 1: Receiver sends $y = g^r h^\alpha, r \in_R Z_q$;
 - 2: Sender sends $c_i = (g^{k_i}, s_i(y/h^i)^{k_i}), k_i \in_R Z_q, 1 \leq i \leq z$;
 - 3: By $c_\alpha = (d, f)$, receiver computes $s_\alpha = f/d^r$
-

SECURE MPC (MULTI-PARTY- COMPUTATION)

Secure multi-party computation

- closely related to the idea of zero-knowledge.
- In general it refers to computational systems in which multiple parties wish to jointly compute some value based on individually held secret bits of information, but do not wish to reveal their secrets to one another in the process.

Secure Computation

- Secure 2-party/multi-party computation:
general-purpose cryptographic protocol.
 - Suppose there are n parties.
 - A common public input: function $f()$.
 - $f()=(f1(),f2())$
 - Each party has a private input x_i .
 - Can we construct a protocol for securely computing function $f(x_1, \dots, x_n)$?
 - A should only learn $f1(x_1, \dots, x_n)$;
 - B should only learn $f2(x_1, \dots, x_n)$.

Adversary Models

- There are two major adversary models for secure computation: Semi-honest model and fully malicious model.
 - **Semi-honest model**: all parties follow the protocol; but dishonest parties may be curious to violate others' privacy.
 - **Fully malicious model**: dishonest parties can deviate from the protocol and behave arbitrarily.
 - Clearly, fully malicious model is harder to deal with.

Security in Semi-Honest Model

- A 2-party protocol between A and B (for computing a **deterministic function** $f()$) is secure in the semi-honest model if there exists an efficient algorithm MA (resp., MB) such that
 - the view of A (resp., B) is **computationally indistinguishable** from $MA(x_1, f_1(x_1, x_2))$ (resp., $MB(x_2, f_2(x_1, x_2))$).
- We can have a similar (but more complex) definition for multiple parties.

Yao's Theorem

- The first completeness theorem for secure computation.
- It states that for **ANY** efficiently computable function, there is a secure two-party protocol in the semi-honest model.
 - Therefore, in theory there is no need to design protocols for specific functions.
 - Surprising!

Circuit Computation

We are now ready to describe the protocol for creating a garbled circuit. Let n be a security parameter, and let $f(x_1, \dots, x_a; y_1, \dots, y_b)$ be a function that Alice and Bob wish to compute, with the x_i representing Alice's inputs and the y_j Bob's. We are of course assuming that f can be run on a computer.

- The design of Yao's protocol is based on circuit computation.
 - Recall any (efficiently) computable function can be represented as a family of (polynomial-size) boolean circuits.
 - Recall such a circuit consists of **and**, or, and **not** gates.
 - It is enough if we can evaluate Alice's private circuit at Bob's private input.

The first thing Alice does to f is *hard-code* her inputs A_1, \dots, A_a into f , creating the function

$$g(y_1, \dots, y_b) = f(A_1, \dots, A_a; y_1, \dots, y_b).$$

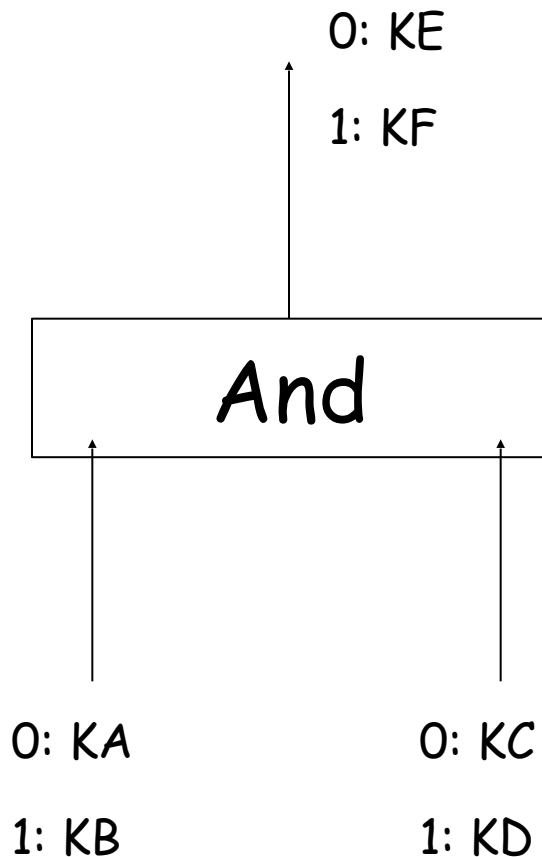
We now want to ensure that Bob cannot intuit what this function does.

We will now obfuscate the circuit using one-time pads. For each wire w_i of the circuit, Alice creates two random keys K_i^0, K_i^1 of length n . The idea is that K_i^0 will correspond to the wire w_i being assigned a value of 0, and likewise for K_i^1 .

Garbled Circuit

- We can represent Alice's circuit with a garbled circuit that does not reveal any knowledge about the circuit.
 - For each edge in the circuit, we use two random keys to represent 0 and 1 respectively.
 - We represent each gate with 4 ciphertexts, for input (0,0), (0,1), (1,0), (1,1), respectively.
 - These ciphertexts should be permuted randomly.
 - The ciphertext for input (a,b) is the key representing the output $\text{Gate}(a,b)$ encrypted by the keys representing a and b.

Example of a Gate



➤ This gate is represented by:
(a random permutation of)

$$E_{KA}(E_{KC}(KE));$$

$$E_{KB}(E_{KC}(KE));$$

$$E_{KA}(E_{KD}(KE));$$

$$E_{KB}(E_{KD}(KF)).$$

Yao's Garbled Gate

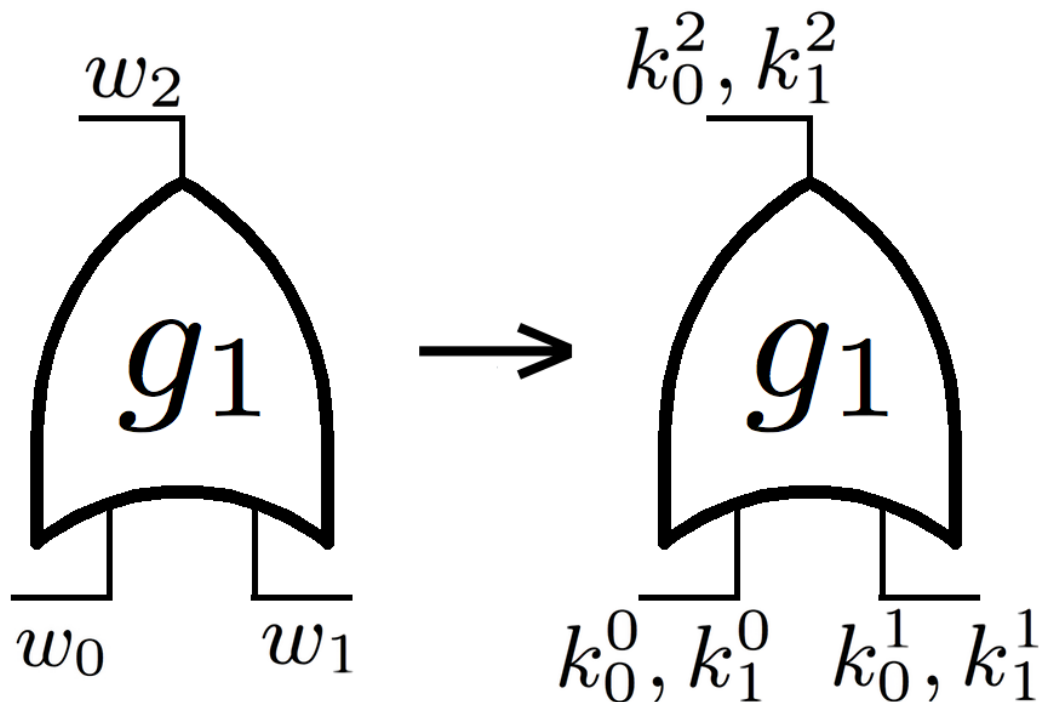


Figure 1: Garbling a single gate

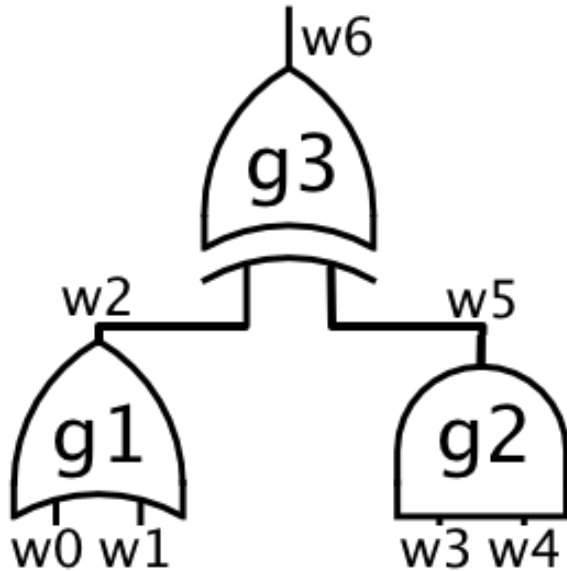
w_0	w_1	w_2
0	0	0
0	1	1
1	0	1
1	1	1

(a) Original Values

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$H(k_0^0 k_1^0 g_1) \oplus k_2^0$
k_0^0	k_1^1	k_2^1	$H(k_0^0 k_1^1 g_1) \oplus k_2^1$
k_0^1	k_1^0	k_2^1	$H(k_0^1 k_1^0 g_1) \oplus k_2^1$
k_0^1	k_1^1	k_2^1	$H(k_0^1 k_1^1 g_1) \oplus k_2^1$

(b) Garbled Values

Several Gates



w_0	w_1	w_2
0	0	0
0	1	1
1	0	1
1	1	1

(a) Original Values

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$H(k_0^0 k_1^0 g_1) \oplus k_2^0$
k_0^0	k_1^1	k_2^1	$H(k_0^0 k_1^1 g_1) \oplus k_2^1$
k_0^1	k_1^0	k_2^1	$H(k_0^1 k_1^0 g_1) \oplus k_2^1$
k_0^1	k_1^1	k_2^1	$H(k_0^1 k_1^1 g_1) \oplus k_2^1$

(b) Garbled Values

Figure 2: Computation table for g_1^{OR}

w_3	w_4	w_5
0	0	0
0	1	0
1	0	0
1	1	1

(a) Original Values

w_3	w_4	w_5	garbled value
k_3^0	k_4^0	k_5^0	$H(k_3^0 k_4^0 g_2) \oplus k_5^0$
k_3^0	k_4^1	k_5^0	$H(k_3^0 k_4^1 g_2) \oplus k_5^0$
k_3^1	k_4^0	k_5^0	$H(k_3^1 k_4^0 g_2) \oplus k_5^0$
k_3^1	k_4^1	k_5^1	$H(k_3^1 k_4^1 g_2) \oplus k_5^1$

(b) Garbled Values

Figure 4: Computation table for g_2^{AND}

w_2	w_5	w_6
0	0	0
0	1	1
1	0	1
1	1	0

(a) Original Values

w_2	w_5	w_6	garbled value
k_2^0	k_5^0	k_6^0	$H(k_2^0 k_5^0 g_3) \oplus k_6^0$
k_2^0	k_5^1	k_6^1	$H(k_2^0 k_5^1 g_3) \oplus k_6^1$
k_2^1	k_5^0	k_6^1	$H(k_2^1 k_5^0 g_3) \oplus k_6^1$
k_2^1	k_5^1	k_6^0	$H(k_2^1 k_5^1 g_3) \oplus k_6^0$

(b) Garbled Values

Figure 5: Computation table for g_3^{XOR}

Evaluation of Garbed Circuit

- Given the two keys (one for Alice and one for Bob) representing the inputs of a gate, we can easily obtain the key representing the output of the gate.
 - Only need to decrypt the corresponding entry using both keys from Alice and Bob
 - But we do not know which entry it is? We can decrypt all entries. Suppose each cleartext contains some redundancy (like a hash value). Then only decryption of the right entry can yield such redundancy.

Translating Input?

- Suppose Alice sends
 - the garbled circuit to Bob.
 - the random keys corresponding to her input bit (for each gate) — a sequence of random k keys for her input k bits
- Then Bob can evaluate the garbled circuit if he knows how to translate each of his K input bit to the corresponding random key.
- Alice cannot send Bob both random keys for these K input bits
 - otherwise, Bob can know more information about the function of the gate (knows at least two corrected values in the truth-table, $\frac{1}{2}$ of the table, instead of only one entry)

Jump Start with Oblivious Transfer

- A solution to this problem is 1-out-of-2 OT for each input bit.
 - Alice sends Bob the corresponding keys representing 0 and 1 using a OT protocol
 - Bob chooses to receive only the key representing his input at this bit.
 - Clearly, Bob can't evaluate the circuit at any other input.

Finishing the Evaluation

- At the end of evaluation, Bob gets the keys representing the output bits of circuit.
 - Alice sends Bob a table of the keys for each output bit.
 - Bob translates the keys back to the output bits.
- For privacy, we need to be careful:
 - The topology of the circuit should be the same for all circuits of a particular input size.
 - Then privacy is guaranteed.

Yao's Protocol Summary

Protocol 2 Yao's Garbled Circuits Protocol

- 1: $P1$ generates a boolean circuit representation c_c of f that takes input i_{P1} from $P1$ and i_{P2} from $P2$.
 - 2: $P1$ transforms c_c by garbling each gate's computation table, creating garbled circuit c_g .
 - 3: $P1$ sends both c_g and the values for the input wires in c_g corresponding to i_{P1} to $P2$.
 - 4: $P2$ uses *1-out-of-2 OT* to receive from $P1$ the garbled values for i_{P2} in c_g .
 - 5: $P2$ calculates c_g with the garbled versions of i_{P1} and i_{P2} and outputs the result.
-

Secure Yao's Protocol for Malicious Users

Protocol 3 Malicious-Secure 1-out-of-2 Oblivious Transfer

- 1: $P1$ has a set of two strings, $S = \{s_0, s_1\}$.
 - 2: $P1$ (sender) and $P2$ (receiver) agree on some q and g such that g is a generator for \mathbb{Z}_q^* .
 - 3: $P1$ selects a value C from \mathbb{Z}_q^* such that $P2$ does not know the discrete log of C in \mathbb{Z}_q^* .
 - 4: $P2$ selects $i \in \{0, 1\}$ corresponding to whether $P2$ wants s_0 or s_1 . $P2$ also selects a random $0 \leq x_i \leq q - 2$.
 - 5: $P2$ sets $\beta_i = g^{x_i}$ and $\beta_{i-1} = C \odot (g^{x_i})^{-1}$. (β_0, β_1) and (i, x_i) form $P1$ public and private keys, respectively.
 - 6: $P1$ checks the validity of $P2$'s public keys by verifying that $\beta_0 \bullet \beta_1 = C$. If not, $P1$ aborts.
 - 7: $P1$ selects y_0, y_1 such that $0 \leq y_0, y_1 \leq q - 2$, and sends $P2$ $a_0 = g^{y_0}$ and $a_1 = g^{y_1}$.
 - 8: $P1$ also generates $z_0 = \beta_0^{y_0}, z_1 = \beta_1^{y_1}$ and sends $P2$ $r_0 = s_0 \oplus z_0$ and $r_1 = s_1 \oplus z_1$.
 - 9: $P2$ computes $z_i = a_i^{x_i}$ and then receives s_i by computing $s_i = z_i \oplus r_i$.
-

Secure Yao's Protocol for Malicious Users

is provided below. Specifically, in step 5 $P1$ checks that $\beta_0 \bullet \beta_1 = C$ to prevent $P2$ from being able to decrypt under both β_0 and β_1 , and to force $P2$ to choose one or the other. As long as the assumption that $P2$ does not know the discrete log of C holds, then it follows that $P2$ cannot know the discrete log of both β_0 and β_1 .

- Second, how to ensure that Alice will only Garble the function claimed to be computed?
 - not the one that could reveal Bob's secret?
 - zero knowledge proof based one is too expensive -- need new design

Secure Against Sender

- ❖ Assure correct encoding of the function f
- ❖ Cut-Paste assures that cheating with prob $1/m$

Protocol 4 Securing Circuit Construction With Cut-and-Choose

- 1: $P1$ generates m garbled versions of the circuit c , along with a corresponding garbled version of his input, called X_i for $0 \leq i < m$.
 - 2: $P1$ uses hash function H to generate commitments to each garbled input, $COM_i = H(X_i)$ for $0 \leq i < m$.
 - 3: $P1$ sends $P2$ m garbled circuits and COM such that $|COM| = m$.
 - 4: $P2$ selects $0 \leq j < m$ and $P1$ un-garbles all circuits except the j th.
 - 5: $P2$ inspects all $m - 1$ circuits to check that they are correctly formed. If not, $P2$ aborts.
 - 6: $P2$ receives $P1$'s garbled inputs to circuit j and confirms that $P1$ did not change his inputs by verifying $COM_j = H(X_j)$. If not, $P2$ aborts.
 - 7: Otherwise, $P2$ receives the continues with Yao's protocol as normal.
-

Secure Against Sender-Improved

❖ Assure correct encoding of the function f

Instead of $P1$ revealing $m - 1$ circuits, Lindell and Pinkas have $P2$ select only $m/2$ circuits to be revealed. $P2$ computes the remaining $m/2$ circuits and takes the majority result. Under this construction, a malicious $P1$ would only succeed in having $P2$ output a corrupt result if

1. $P1$ constructs more than $m/4$ of the circuits corruptly, and
2. None of the corrupt $m/4$ circuits are among the $m/2$ circuits $P2$ selected to be revealed.

Lindell and Pinkas measure $P1$'s chance of success in such an attack at $2^{-0.311m}$, where m is the number of circuits generated[16].

Other Attacks

- ❖ Previous method assures that
 - ❖ Bob will not get TWO random keys (secure OT protocol)
 - ❖ Alice will send Bob the corrected garbled circuit
- ❖ How to assure that Alice will send Bob the corrected random keys in the 1-out-2 OT protocol?
 - ❖ if Alice sends correct key for Bob's bit 0, but corrupted key for Bob's bit 1
 - ❖ Then Bob is able to compute correct value if Bob's input is 0, otherwise, Bob will abort
 - ❖ Thus Alice learns Bob's bit!

Secure Against Alice in OT

figure 7. The defense works by adding $s|i_{P2}|$ input bits to the circuit, where s is a chosen security parameter. Each of $P2$'s input bits is replaced by the result of XOR of s new input bits, each chosen by $P2$ from $P1$ through the same OT protocol. The circuit is also augmented to reflect these new input wires and XOR gates. This step of indirection gives $P2$ 2^{s-1} ways to receive her true input bits from $P1$, and prevents $P1$ from gaining any knowledge about $P2$'s underlying input bit by corrupting the augmented, XORed inputs.

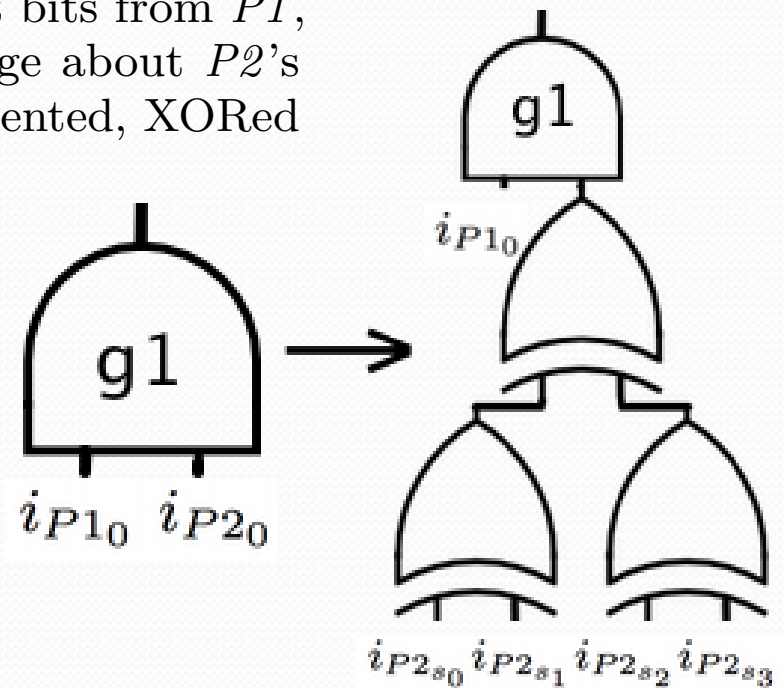


Figure 7: Securing against $P1$ providing malicious inputs through s new \oplus gates

How to ensure that Output is sent

❖ sent from user $P2$ to user $P1$?

One remaining problem with Yao's protocol in the *malicious* setting is how to ensure $P2$ returns to $P1$ any output value from the function. There is no clear method to prevent $P2$ from maliciously aborting the protocol early, right after computing the output of the garbled circuit, but right before sending the output to $P1$. That this problem is still open means that the *fairness* principle Yao described for SFE cannot be guaranteed in the *malicious* setting.

How to ensure Efficiency?

❖ Computation and Communication?

Kreuter, Shelat and Shen[14] found that computing the edit distance of two 4095-bit strings required a circuit of over 5.9 billion gates and several hours of time, even with a highly optimized circuit.

A great deal of work has been done to make Yao's protocol less expensive to execute. This work broadly falls into three categories: 1) communication optimizations that reduce the amount of information that must be shared between the two parties, 2) execution optimizations that allow for the same number of gates to be executed in a shorter amount of time, and 3) circuit optimizations that reducing the number of gates needed to compute a function.

Some techniques for reducing communication

- ❖ Use random seed to generate ALL the random keys for Alice's Circuit Input (not Bob?)
 - ❖ Then both Alice and Bob can generate keys without communicating
- ❖ only sending the hashed version of the Garbled Circuits first
 - ❖ then Bob choose $m/2$ randomly to reveal
 - ❖ together with random seed for generating the input random keys for Alice
- ❖ Total reduction by $\frac{1}{2}$

Implementations

- ❖ Fairplay 2004
- ❖ Huang, Evans, Katz, Malka 2011
- ❖ Kreuter, Shelat, Shen 2012

End

Feasibility - A Fundamental Theorem

- Any multiparty functionality can be securely computed
 - For any number of corrupted parties: security with abort is achieved, assuming enhanced trapdoor permutations [Yao,GMW]
 - With an honest majority: full security is achieved, assume private channels only [BGW,CCD]
- These results hold in the **stand-alone model**
 - When composition is considered, things are much more difficult

Application to Private Data Mining

- **The setting:**
 - Data is **distributed** at different sites
 - These sites may be third parties (e.g., hospitals, government bodies) or may be the individual him or herself
- **The aim:**
 - Compute the data mining algorithm on the data so that **nothing but the output is learned**
 - That is, carry out a **secure computation**
- **Conclusion:** private data mining is solved **in principle**

Privacy \neq Security

- As we have mentioned, secure computation only deals with the process of computing the function
 - It does not ask whether or not the function should be computed

Privacy and Secure Computation

- Secure computation can be used to solve any distributed data-mining problem
- A two-stage process:
 - Decide that the function / algorithm should be computed – an issue of **privacy**
 - Apply secure computation techniques to compute it securely – **security**
- But, not every privacy problem can be cast as a distributed computation