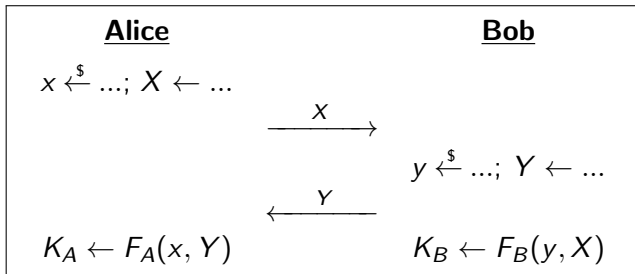


COMPUTATIONAL NUMBER THEORY

Secret key exchange

Problem: Obtain a joint secret key via interaction over a public channel:



Desired properties of the protocol:

- $K_A = K_B$, meaning Alice and Bob agree on a key
- Adversary given X, Y can't compute K_A

Secret Key Exchange

Can you build a secret key exchange protocol?

Secret Key Exchange

Can you build a secret key exchange protocol?

Symmetric cryptography has existed for thousands of years.

But no secret key exchange protocol was found in that time.

Many people thought it was impossible.

Secret Key Exchange

Can you build a secret key exchange protocol?

Symmetric cryptography has existed for thousands of years.

But no secret key exchange protocol was found in that time.

Many people thought it was impossible.

In 1976, Diffie and Hellman proposed one.

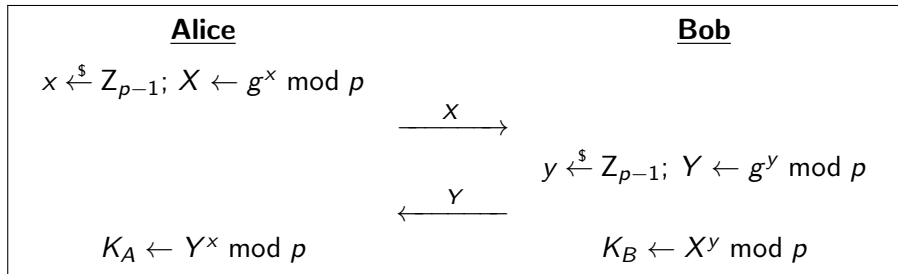
This was the birth of public-key (asymmetric) cryptography.

DH Key Exchange Video

<http://www.youtube.com/watch?v=3QnD2c4Xovk>

DH Secret Key Exchange

The following are assumed to be public: A large prime p and a number g called a generator mod p . Let $Z_{p-1} = \{0, 1, \dots, p-2\}$.



- $Y^x = (g^y)^x = g^{xy} = (g^x)^y = X^y$ modulo p , so $K_A = K_B$
- Adversary is faced with computing $g^{xy} \text{ mod } p$ given $g^x \text{ mod } p$ and $g^y \text{ mod } p$, which nobody knows how to do efficiently for large p .

DH Secret Key Exchange: Questions

- How do we pick a large prime p , and how large is large enough?
- What does it mean for g to be a generator modulo p ?
- How do we find a generator modulo p ?
- How can Alice quickly compute $x \mapsto g^x \bmod p$?
- How can Bob quickly compute $y \mapsto g^y \bmod p$?
- Why is it hard to compute $(g^x \bmod p, g^y \bmod p) \mapsto g^{xy} \bmod p$?
- ...

To answer all that and more, we will forget about DH secret key exchange for a while and take a trip into computational number theory ...

Notation

$$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$$

$$\mathbb{N} = \{0, 1, 2, \dots\}$$

$$\mathbb{Z}_+ = \{1, 2, 3, \dots\}$$

For $a, N \in \mathbb{Z}$ let $\gcd(a, N)$ be the largest $d \in \mathbb{Z}_+$ such that d divides both a and N .

Example: $\gcd(30, 70) = 10$.

Integers mod N

For $N \in \mathbb{Z}_+$, let

- $\mathbb{Z}_N = \{0, 1, \dots, N - 1\}$
- $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$
- $\varphi(N) = |\mathbb{Z}_N^*|$

Example: $N = 12$

- $\mathbb{Z}_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$
- $\mathbb{Z}_{12}^* =$

Integers mod N

For $N \in \mathbb{Z}_+$, let

- $\mathbb{Z}_N = \{0, 1, \dots, N - 1\}$
- $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$
- $\varphi(N) = |\mathbb{Z}_N^*|$

Example: $N = 12$

- $\mathbb{Z}_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$
- $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$
- $\varphi(12) =$

For $N \in \mathbb{Z}_+$, let

- $\mathbb{Z}_N = \{0, 1, \dots, N - 1\}$
- $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$
- $\varphi(N) = |\mathbb{Z}_N^*|$

Example: $N = 12$

- $\mathbb{Z}_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$
- $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$
- $\varphi(12) = 4$

Division and mod

INT-DIV(a, N) returns (q, r) such that

- $a = qN + r$
- $0 \leq r < N$

Refer to q as the **quotient** and r as the **remainder**. Then

$$a \bmod N = r \in \mathbb{Z}_N$$

is the remainder when a is divided by N .

Example: INT-DIV(17, 3) = (5, 2) and $17 \bmod 3 = 2$.

Def: $a \equiv b \pmod{N}$ if $a \bmod N = b \bmod N$.

Example: $17 \equiv 14 \pmod{3}$

Let G be a non-empty set, and let \cdot be a binary operation on G . This means that for every two points $a, b \in G$, a value $a \cdot b$ is defined.

Example: $G = \mathbb{Z}_{12}^*$ and “ \cdot ” is multiplication modulo 12, meaning

$$a \cdot b = ab \bmod 12$$

Def: We say that G is a *group* if it has four properties called closure, associativity, identity and inverse that we present next.

Fact: If $N \in \mathbb{Z}_+$ then $G = \mathbb{Z}_N^*$ with $a \cdot b = ab \bmod N$ is a group.

Closure: For every $a, b \in G$ we have $a \cdot b$ is also in G .

Example: $G = \mathbb{Z}_{12}$ with $a \cdot b = ab$ does not have closure because $7 \cdot 5 = 35 \notin \mathbb{Z}_{12}$.

Fact: If $N \in \mathbb{Z}_+$ then $G = \mathbb{Z}_N^*$ with $a \cdot b = ab \bmod N$ satisfies closure, meaning

$$\gcd(a, N) = \gcd(b, N) = 1 \text{ implies } \gcd(ab \bmod N, N) = 1$$

Example: Let $G = \mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$. Then

$$5 \cdot 7 \bmod 12 = 35 \bmod 12 = 11 \in \mathbb{Z}_{12}^*$$

Groups: Associativity

Associativity: For every $a, b, c \in G$ we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

Fact: If $N \in \mathbb{Z}_+$ then $G = \mathbb{Z}_N^*$ with $a \cdot b = ab \bmod N$ satisfies associativity, meaning

$$((ab \bmod N)c) \bmod N = (a(bc \bmod N)) \bmod N$$

Example:

$$\begin{aligned}(5 \cdot 7 \bmod 12) \cdot 11 \bmod 12 &= (35 \bmod 12) \cdot 11 \bmod 12 \\ &= 11 \cdot 11 \bmod 12 = 1\end{aligned}$$

$$\begin{aligned}5 \cdot (7 \cdot 11 \bmod 12) \bmod 12 &= 5 \cdot (77 \bmod 12) \bmod 12 \\ &= 5 \cdot 5 \bmod 12 = 1\end{aligned}$$

Identity element: There exists an element $1 \in G$ such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$.

Fact: If $N \in \mathbb{Z}_+$ and $G = \mathbb{Z}_N^*$ with $a \cdot b = ab \bmod N$ then 1 is the identity element because $a \cdot 1 \bmod N = 1 \cdot a \bmod N = a$ for all a .

Inverses: For every $a \in G$ there exists a unique $b \in G$ such that $a \cdot b = b \cdot a = 1$.

This b is called the inverse of a and is denoted a^{-1} if G is understood.

Fact: If $N \in \mathbb{Z}_+$ and $G = \mathbb{Z}_N^*$ with $a \cdot b = ab \bmod N$ then $\forall a \in \mathbb{Z}_N^* \exists b \in \mathbb{Z}_N^*$ such that $a \cdot b \bmod N = 1$.

We denote this unique inverse b by $a^{-1} \bmod N$.

Example: $5^{-1} \bmod 12$ is the $b \in \mathbb{Z}_{12}^*$ satisfying $5b \bmod 12 = 1$, so $b =$

Inverses: For every $a \in G$ there exists a unique $b \in G$ such that $a \cdot b = b \cdot a = 1$.

This b is called the inverse of a and is denoted a^{-1} if G is understood.

Fact: If $N \in \mathbb{Z}_+$ and $G = \mathbb{Z}_N^*$ with $a \cdot b = ab \bmod N$ then $\forall a \in \mathbb{Z}_N^* \exists b \in \mathbb{Z}_N^*$ such that $a \cdot b \bmod N = 1$.

We denote this unique inverse b by $a^{-1} \bmod N$.

Example: $5^{-1} \bmod 12$ is the $b \in \mathbb{Z}_{12}^*$ satisfying $5b \bmod 12 = 1$, so $b = 5$

Examples of groups

Fact: If $N \geq 1$ is an integer then Z_N is a group under the operation of addition modulo N , namely $a \cdot b = (a + b) \bmod N$.

Fact: If $N \geq 2$ is an integer then Z_N^* is a group under the operation of multiplication modulo N , namely $a \cdot b = (ab) \bmod N$.

Computational Shortcuts

Fact: Let $a, b, c \in \mathbb{Z}$ and $N \in \mathbb{Z}_+$. Then

$$abc \bmod N = ((ab \bmod N) c) \bmod N$$

Example: What is $5 \cdot 8 \cdot 10 \cdot 16 \bmod 21$?

Slow way:

- $5 \cdot 8 \cdot 10 \cdot 16 = 40 \cdot 10 \cdot 16 = 400 \cdot 16 = 6400$
- $6400 \bmod 21 = 16$

Faster way, using above Fact:

- $5 \cdot 8 \bmod 21 = 40 \bmod 21 = 19$
- $19 \cdot 10 \bmod 21 = 190 \bmod 21 = 1$
- $1 \cdot 16 \bmod 21 = 16$

Exponentiation

Let G be a group and $a \in G$. We let $a^0 = 1$ be the **identity** element and for $n \geq 1$, we let

$$a^n = \underbrace{a \cdot a \cdots a}_n.$$

Also we let

$$a^{-n} = \underbrace{a^{-1} \cdot a^{-1} \cdots a^{-1}}_n.$$

This ensures that for all $i, j \in \mathbb{Z}$,

- $a^{i+j} = a^i \cdot a^j$
- $a^{ij} = (a^i)^j = (a^j)^i$
- $a^{-i} = (a^i)^{-1} = (a^{-1})^i$

Meaning we can manipulate exponents “as usual”.

Examples

Let $N = 14$ and $G = \mathbb{Z}_N^*$. Then modulo N we have

$$5^3 =$$

Examples

Let $N = 14$ and $G = \mathbb{Z}_N^*$. Then modulo N we have

$$5^3 = 5 \cdot 5 \cdot 5$$

Examples

Let $N = 14$ and $G = \mathbb{Z}_N^*$. Then modulo N we have

$$5^3 = 5 \cdot 5 \cdot 5 \equiv 25 \cdot 5 \equiv 11 \cdot 5 \equiv 55 \equiv 13$$

and

$$5^{-3} =$$

Examples

Let $N = 14$ and $G = \mathbb{Z}_N^*$. Then modulo N we have

$$5^3 = 5 \cdot 5 \cdot 5 \equiv 25 \cdot 5 \equiv 11 \cdot 5 \equiv 55 \equiv 13$$

and

$$5^{-3} = 5^{-1} \cdot 5^{-1} \cdot 5^{-1}$$

Examples

Let $N = 14$ and $G = \mathbb{Z}_N^*$. Then modulo N we have

$$5^3 = 5 \cdot 5 \cdot 5 \equiv 25 \cdot 5 \equiv 11 \cdot 5 \equiv 55 \equiv 13$$

and

$$5^{-3} = 5^{-1} \cdot 5^{-1} \cdot 5^{-1} \equiv 3 \cdot 3 \cdot 3$$

Examples

Let $N = 14$ and $G = \mathbb{Z}_N^*$. Then modulo N we have

$$5^3 = 5 \cdot 5 \cdot 5 \equiv 25 \cdot 5 \equiv 11 \cdot 5 \equiv 55 \equiv 13$$

and

$$5^{-3} = 5^{-1} \cdot 5^{-1} \cdot 5^{-1} \equiv 3 \cdot 3 \cdot 3 \equiv 27 \equiv 13$$

Group Orders

The **order** of a group G is its size $|G|$, meaning the number of elements in it.

Example: The order of Z_{21}^* is

Group Orders

The **order** of a group G is its size $|G|$, meaning the number of elements in it.

Example: The order of Z_{21}^* is 12 because

$$Z_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$

Fact: Let G be a group of order m and $a \in G$. Then, $a^m = 1$.

Examples: Modulo 21 we have

- $5^{12} \equiv (5^3)^4 \equiv 20^4 \equiv (-1)^4 \equiv 1$
- $8^{12} \equiv (8^2)^6 \equiv (1)^6 \equiv 1$

Simplifying exponentiation

Fact: Let G be a group of order m and $a \in G$. Then, $a^m = 1$.

Corollary: Let G be a group of order m and $a \in G$. Then for any $i \in \mathbb{Z}$,

$$a^i = a^{i \bmod m}.$$

Proof: Let $(q, r) \leftarrow \text{INT-DIV}(i, m)$, so that $i = mq + r$ and $r = i \bmod m$.
Then

$$a^i = a^{mq+r} = (a^m)^q \cdot a^r$$

But $a^m = 1$ by Fact.

Simplifying exponentiation

Corollary: Let G be a group of order m and $a \in G$. Then for any $i \in \mathbb{Z}$,

$$a^i = a^{i \bmod m}.$$

Example: What is $5^{74} \bmod 21$?

Simplifying exponentiation

Corollary: Let G be a group of order m and $a \in G$. Then for any $i \in \mathbb{Z}$,

$$a^i = a^{i \bmod m}.$$

Example: What is $5^{74} \bmod 21$?

Solution: Let $G = \mathbb{Z}_{21}^*$ and $a = 5$. Then, $m = 12$, so

$$\begin{aligned} 5^{74} \bmod 21 &= 5^{74 \bmod 12} \bmod 21 \\ &= 5^2 \bmod 21 \\ &= 4. \end{aligned}$$

Measuring Running Time of Algorithms on Numbers

In an algorithms course, the cost of arithmetic is often assumed to be $\mathcal{O}(1)$, because numbers are small. In cryptography numbers are

very, very BIG!

Typical sizes are 2^{512} , 2^{1024} , 2^{2048} .

Numbers are provided to algorithms in binary. The length of a , denoted $|a|$, is the number of bits in the binary encoding of a .

Example: $|7| = 3$ because 7 is 111 in binary.

Running time is measured as a function of the lengths of the inputs.

Algorithms on numbers

Algorithm	Input	Output	Time
ADD	a, b	$a + b$	$\mathcal{O}(a + b)$
MULT	a, b	ab	$\mathcal{O}(a \cdot b)$
INT-DIV	a, N	q, r	$\mathcal{O}(a \cdot N)$
MOD	a, N	$a \bmod N$	$\mathcal{O}(a \cdot N)$
EXT-GCD	a, N	(d, a', N')	$\mathcal{O}(a \cdot N)$
MOD-INV	$a \in \mathbb{Z}_N^*, N$	$a^{-1} \bmod N$	$\mathcal{O}(N ^2)$
MOD-EXP	$a \in \mathbb{Z}_N, n, N$	$a^n \bmod N$	$\mathcal{O}(n \cdot N ^2)$
EXP _G	$a \in G, n$	$a^n \in G$	$\mathcal{O}(n)$ G-ops

EXT-GCD(a, N) returns (d, a', N') such that

$$d = \gcd(a, N) = a \cdot a' + N \cdot N' .$$

Example: EXT-GCD(12, 20) =

EXT-GCD(a, N) returns (d, a', N') such that

$$d = \gcd(a, N) = a \cdot a' + N \cdot N' .$$

Example: EXT-GCD(12, 20) = (4, -3, 2) because

$$4 = \gcd(12, 20) = 12 \cdot (-3) + 20 \cdot 2 .$$

Extended gcd Algorithm

EXT-GCD(a, N) \mapsto (d, a', N') such that

$$d = \gcd(a, N) = a \cdot a' + N \cdot N' .$$

Lemma: Let $(q, r) = \text{INT-DIV}(a, N)$. Then, $\gcd(a, N) = \gcd(N, r)$

Alg EXT-GCD(a, N) // (a, N) \neq (0, 0)

if $N = 0$ then return ($a, 1, 0$)

else

$(q, r) \leftarrow \text{INT-DIV}(a, N)$; $(d, x, y) \leftarrow \text{EXT-GCD}(N, r)$

$a' \leftarrow y$; $N' \leftarrow x - qy$; return (d, a', N')

Running time is $\mathcal{O}(|a| \cdot |N|)$, so the extended gcd can be computed in **quadratic** time. If $a \geq N > 0$ then $\text{abs}(a') \leq N$ and $\text{abs}(N') \leq a$ where $\text{abs}(\cdot)$ denotes the absolute value. Analysis showing all this is non-trivial (worst case is Fibonacci numbers).

Modular Inverse

For a, N such that $\gcd(a, N) = 1$, we want to compute $a^{-1} \pmod N$, meaning the unique $a' \in \mathbb{Z}_N^*$ satisfying $aa' \equiv 1 \pmod N$.

But if we let $(d, a', N') \leftarrow \text{EXT-GCD}(a, N)$ then

$$d = 1 = \gcd(a, N) = a \cdot a' + N \cdot N'$$

But $N \cdot N' \equiv 0 \pmod N$ so $aa' \equiv 1 \pmod N$

Alg MOD-INV(a, N)

$(d, a', N') \leftarrow \text{EXT-GCD}(a, N)$

return $a' \pmod N$

Modular inverse can be computed in **quadratic** time.

Modular Exponentiation

Let G be a group and $a \in G$. For $n \in \mathbb{N}$, we want to compute $a^n \in G$.

We know that

$$a^n = \underbrace{a \cdot a \cdots a}_n$$

Consider:

```
y ← 1
for i = 1, ..., n do y ← y · a
return y
```

Question: Is this a good algorithm?

Modular Exponentiation

Let G be a group and $a \in G$. For $n \in \mathbb{N}$, we want to compute $a^n \in G$.

We know that

$$a^n = \underbrace{a \cdot a \cdots a}_n$$

Consider:

```
y ← 1
for i = 1, ..., n do y ← y · a
return y
```

Question: Is this a good algorithm?

Answer: It is correct but **VERY SLOW**. The number of group operations is $\mathcal{O}(n) = \mathcal{O}(2^{|n|})$ so it is exponential time. For $n \approx 2^{512}$ it is prohibitively expensive.

Fast exponentiation idea

We can compute

$$a \longrightarrow a^2 \longrightarrow a^4 \longrightarrow a^8 \longrightarrow a^{16} \longrightarrow a^{32}$$

in just 5 steps by repeated squaring. So we can compute a^n in i steps when $n = 2^i$.

But what if n is not a power of 2?

Square-and-Multiply Exponentiation Example

Suppose the binary length of n is 5, meaning the binary representation of n has the form $b_4b_3b_2b_1b_0$. Then

$$\begin{aligned}n &= 2^4b_4 + 2^3b_3 + 2^2b_2 + 2^1b_1 + 2^0b_0 \\ &= 16b_4 + 8b_3 + 4b_2 + 2b_1 + b_0 .\end{aligned}$$

We want to compute a^n . Our exponentiation algorithm will proceed to compute the values $y_5, y_4, y_3, y_2, y_1, y_0$ in turn, as follows:

$$\begin{aligned}y_5 &= 1 \\ y_4 &= y_5^2 \cdot a^{b_4} = a^{b_4} \\ y_3 &= y_4^2 \cdot a^{b_3} = a^{2b_4+b_3} \\ y_2 &= y_3^2 \cdot a^{b_2} = a^{4b_4+2b_3+b_2} \\ y_1 &= y_2^2 \cdot a^{b_1} = a^{8b_4+4b_3+2b_2+b_1} \\ y_0 &= y_1^2 \cdot a^{b_0} = a^{16b_4+8b_3+4b_2+2b_1+b_0} .\end{aligned}$$

Square-and-Multiply Exponentiation Algorithm

Let $\text{bin}(n) = b_{k-1} \dots b_0$ be the binary representation of n , meaning

$$n = \sum_{i=0}^{k-1} b_i 2^i$$

Alg $\text{EXP}_G(a, n)$ // $a \in G, n \geq 1$
 $b_{k-1} \dots b_0 \leftarrow \text{bin}(n)$
 $y \leftarrow 1$
for $i = k - 1$ downto 0 do $y \leftarrow y^2 \cdot a^{b_i}$
return y

The running time is $\mathcal{O}(|n|)$ group operations.

$\text{MOD-EXP}(a, n, N)$ returns $a^n \bmod N$ in time $\mathcal{O}(|n| \cdot |N|^2)$, meaning is **cubic** time.

Algorithms on numbers

Algorithm	Input	Output	Time
ADD	a, b	$a + b$	$\mathcal{O}(a + b)$
MULT	a, b	ab	$\mathcal{O}(a \cdot b)$
INT-DIV	a, N	q, r	$\mathcal{O}(a \cdot N)$
MOD	a, N	$a \bmod N$	$\mathcal{O}(a \cdot N)$
EXT-GCD	a, N	(d, a', N')	$\mathcal{O}(a \cdot N)$
MOD-INV	$a \in \mathbb{Z}_N^*, N$	$a^{-1} \bmod N$	$\mathcal{O}(N ^2)$
MOD-EXP	$a \in \mathbb{Z}_N, n, N$	$a^n \bmod N$	$\mathcal{O}(n \cdot N ^2)$
EXP_G	$a \in G, n$	$a^n \in G$	$\mathcal{O}(n)$ G-ops

Generators and cyclic groups

Let G be a group of order m and let $g \in G$. We let

$$\langle g \rangle = \{ g^i : i \in \mathbb{Z}_m \} .$$

The size $|\langle g \rangle|$ of the set $\langle g \rangle$ need not equal m . It could be smaller. It is always a divisor of m .

The *order* of g is defined to be $|\langle g \rangle|$.

We say that $g \in G$ is a *generator* (or primitive element) of G if $\langle g \rangle = G$, meaning the order of g is m .

We say that G is *cyclic* if it has a generator, meaning there exists $g \in G$ such that g is a generator of G .

Generators and cyclic groups: Example

Let $G = Z_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, which has order $m = 10$.

i	0	1	2	3	4	5	6	7	8	9	10
$2^i \bmod 11$	1	2	4	8	5	10	9	7	3	6	1
$5^i \bmod 11$	1	5	3	4	9	1	5	3	4	9	1

so

$$\langle 2 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$\langle 5 \rangle = \{1, 3, 4, 5, 9\}$$

- 2 a generator because $\langle 2 \rangle = Z_{11}^*$.
- 5 is not a generator because $\langle 5 \rangle \neq Z_{11}^*$.
- Z_{11}^* is cyclic because it has a generator.

If $G = \langle g \rangle$ is a cyclic group of order m then for every $a \in G$ there is a **unique** exponent $i \in \mathbb{Z}_m$ such that $g^i = a$. We call i the discrete logarithm of a to base g and denote it by

$$\text{DLog}_{G,g}(a)$$

The discrete log function is the inverse of the exponentiation function:

$$\begin{aligned} \text{DLog}_{G,g}(g^i) &= i \quad \text{for all } i \in \mathbb{Z}_m \\ g^{\text{DLog}_{G,g}(a)} &= a \quad \text{for all } a \in G. \end{aligned}$$

Discrete Logarithms: Example

Let $G = \mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, which is a cyclic group of order $m = 10$. We know that 2 is a generator, so $\text{DLog}_{G,2}(a)$ is the exponent $i \in \mathbb{Z}_{10}$ such that $2^i \bmod 11 = a$.

i	0	1	2	3	4	5	6	7	8	9
$2^i \bmod 11$	1	2	4	8	5	10	9	7	3	6

a	1	2	3	4	5	6	7	8	9	10
$\text{DLog}_{G,2}(a)$										

Discrete Logarithms: Example

Let $G = \mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, which is a cyclic group of order $m = 10$. We know that 2 is a generator, so $\text{DLog}_{G,2}(a)$ is the exponent $i \in \mathbb{Z}_{10}$ such that $2^i \bmod 11 = a$.

i	0	1	2	3	4	5	6	7	8	9
$2^i \bmod 11$	1	2	4	8	5	10	9	7	3	6

a	1	2	3	4	5	6	7	8	9	10
$\text{DLog}_{G,2}(a)$	0	1	8	2	4	9	7	3	6	5

Finding Cyclic Groups

Fact 1: Let p be a prime. Then Z_p^* is cyclic.

Fact 2: Let G be any group whose order $m = |G|$ is a prime number. Then G is cyclic.

Note: $|Z_p^*| = p - 1$ is **not** prime, so **Fact 2** doesn't imply **Fact 1**.

Computing Discrete Logs

Let $G = \langle g \rangle$ be a cyclic group of order m with generator $g \in G$.

Input: $X \in G$

Desired Output: $\text{DLog}_{G,g}(X)$

That is, we want x such that $g^x = X$.

for $x = 0, \dots, m - 1$ do

 if $g^x = X$ then return x

Is this a good algorithm?

Computing Discrete Logs

Let $G = \langle g \rangle$ be a cyclic group of order m with generator $g \in G$.

Input: $X \in G$

Desired Output: $\text{DLog}_{G,g}(X)$

That is, we want x such that $g^x = X$.

for $x = 0, \dots, m - 1$ do

 if $g^x = X$ then return x

Is this a good algorithm? It is

- Correct (always returns the right answer)

Computing Discrete Logs

Let $G = \langle g \rangle$ be a cyclic group of order m with generator $g \in G$.

Input: $X \in G$

Desired Output: $\text{DLog}_{G,g}(X)$

That is, we want x such that $g^x = X$.

for $x = 0, \dots, m - 1$ do

 if $g^x = X$ then return x

Is this a good algorithm? It is

- Correct (always returns the right answer), but
- SLOW!

Run time is $O(m)$ exponentiations, which for $G = \mathbb{Z}_p^*$ is $O(p)$, which is exponential time and prohibitive for large p .

Computing Discrete Logs: Best known algorithms

Group	Time to find discrete logarithms
Z_p^*	$e^{1.92(\ln p)^{1/3}(\ln \ln p)^{2/3}}$
EC_p	$\sqrt{p} = e^{\ln(p)/2}$

Here p is a prime and EC_p represents an elliptic curve group of order p .

In the first case, if the largest factor of $p - 1$ is q , there is also a $O(\sqrt{q})$ algorithm to solve discrete log.

In neither case is a polynomial-time algorithm known.

This (apparent, conjectured) computational intractability of the discrete log problem makes it the basis for cryptographic schemes in which breaking the scheme requires discrete log computation.

Discrete logarithm computation records

In Z_p^* :

$ p $ in bits	When
431	2005
530	2007
596	2014
795	2019

For elliptic curves, current record seems to be for $|p|$ around 114.

Elliptic curve groups

Elliptic curve groups are commonly used for public-key cryptography now.

The mathematical details are a bit complex.

For now, think of an elliptic curve group as a cyclic group.

This means it has a generator, a group operation (typically written as $+$), an order, and one can define the analogue of discrete logarithm in this group.

The structure of elliptic curve groups does not seem to permit the same types of subexponential-time discrete logarithm algorithms as Z_p^* .

Why Elliptic curve (EC) groups?

Say we want 80-bits of security, meaning discrete log computation by the best known algorithm should take time 2^{80} . Then

- If we work in Z_p^* (p a prime) we need to set $|Z_p^*| = p - 1 \approx 2^{1024}$
- But if we work on an elliptic curve group of prime order p then it suffices to set $p \approx 2^{160}$.

This is because

$$e^{1.92(\ln 2^{1024})^{1/3}(\ln \ln 2^{1024})^{2/3}} \approx \sqrt{2^{160}} = 2^{80}$$

But now:

Group Size	Cost of Exponentiation
2^{160}	$T \approx 160^3$
2^{1024}	$1024^3 \approx 260T$

Exponentiation will be 260 times faster in the smaller group.

Let $G = \langle g \rangle$ be a cyclic group of order m , and A an adversary.

Game $DL_{G,g}$

procedure Initialize

$x \xleftarrow{\$} Z_m; X \leftarrow g^x$

return X

procedure Finalize(x')

return $(x = x')$

The **dl-advantage** of A is

$$\text{Adv}_{G,g}^{\text{dl}}(A) = \Pr \left[DL_{G,g}^A \Rightarrow \text{true} \right]$$

CDH: The Computational Diffie-Hellman Problem

Let $G = \langle g \rangle$ be a cyclic group of order m with generator $g \in G$. The CDH problem is:

Input: $X = g^x \in G$ and $Y = g^y \in G$

Desired Output: $g^{xy} \in G$

This underlies security of the DH Secret Key Exchange Protocol.

Obvious algorithm: $x \leftarrow \text{DLog}_{G,g}(X)$; Return Y^x .

So if one can compute discrete logarithms then one can solve the CDH problem.

The converse is an open question. Potentially, there is a way to quickly solve CDH that avoids computing discrete logarithms. But no such way is known.

CDH Formally

Let $G = \langle g \rangle$ be a cyclic group of order m , and A an adversary.

Game $\text{CDH}_{G,g}$

procedure Initialize

$x, y \xleftarrow{\$} Z_m$

$X \leftarrow g^x; Y \leftarrow g^y$

return X, Y

procedure Finalize(Z)

return $(Z = g^{xy})$

The **cdh-advantage** of A is

$$\text{Adv}_{G,g}^{\text{cdh}}(A) = \Pr \left[\text{CDH}_{G,g}^A \Rightarrow \text{true} \right]$$

Building cyclic groups

We will need to build (large) groups over which our cryptographic schemes can work, and find generators in these groups.

How do we do this efficiently?

Building cyclic groups

To find a suitable prime p and generator g of Z_p^* :

- Pick numbers p at random until p is a prime of the desired form
- Pick elements g from Z_p^* at random until g is a generator

For this to work we need to know

- How to test if p is prime
- How many numbers in a given range are primes of the desired form
- How to test if g is a generator of Z_p^* when p is prime
- How many elements of Z_p^* are generators

Finding primes

Desired: An efficient algorithm that given an integer k returns a prime $p \in \{2^{k-1}, \dots, 2^k - 1\}$ such that $q = (p - 1)/2$ is also prime.

Alg Findprime(k)

do

$p \leftarrow^{\$} \{2^{k-1}, \dots, 2^k - 1\}$

until (p is prime and $(p - 1)/2$ is prime)

return p

- How do we test primality?
- How many iterations do we need to succeed?

Primality Testing

Given: integer N

Output: TRUE if N is prime, FALSE otherwise.

```
for  $i = 2, \dots, \lceil \sqrt{N} \rceil$  do
  if  $N \bmod i = 0$  then return false
return true
```

Primality Testing

Given: integer N

Output: TRUE if N is prime, FALSE otherwise.

```
for  $i = 2, \dots, \lceil \sqrt{N} \rceil$  do
  if  $N \bmod i = 0$  then return false
return true
```

Correct but SLOW! $O(N)$ running time, exponential. However, we have:

- $O(|N|^3)$ time randomized algorithms
- Even a $O(|N|^8)$ time deterministic algorithm

Density of primes

Let $\pi(N)$ be the number of primes in the range $1, \dots, N$. So if $p \leftarrow^{\$} \{1, \dots, N\}$ then

$$\Pr [p \text{ is a prime}] = \frac{\pi(N)}{N}$$

Fact: $\pi(N) \sim \frac{N}{\ln(N)}$

So

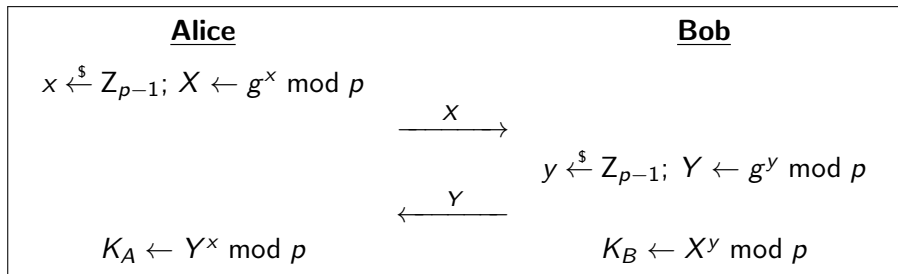
$$\Pr [p \text{ is a prime}] \sim \frac{1}{\ln(N)}$$

If $N = 2^{1024}$ this is about $0.001488 \approx 1/1000$.

So the number of iterations taken by our algorithm to find a prime is not too big.

Recall DH Secret Key Exchange

The following are assumed to be public: A large prime p and a generator g of Z_p^* .



- $Y^x = (g^y)^x = g^{xy} = (g^x)^y = X^y$ modulo p , so $K_A = K_B$
- Adversary is faced with the CDH problem.

DH Secret Key Exchange: Questions

- How do we pick a large prime p , and how large is large enough?
- What does it mean for g to be a generator modulo p ?
- How do we find a generator modulo p ?
- How can Alice quickly compute $x \mapsto g^x \bmod p$?
- How can Bob quickly compute $y \mapsto g^y \bmod p$?
- Why is it hard to compute $(g^x \bmod p, g^y \bmod p) \mapsto g^{xy} \bmod p$?
- ...

The slides have sketched the answers to many of these questions.

Recall that $\varphi(N) = |Z_N^*|$.

Claim: Suppose $e, d \in Z_{\varphi(N)}^*$ satisfy $ed \bmod \varphi(N) = 1$. Then for any $x \in Z_N^*$ we have

$$(x^e)^d \bmod N = x .$$

Proof:

$$\begin{aligned}(x^e)^d \bmod N &= x^{ed \bmod \varphi(N)} \bmod N \\ &= x^1 \bmod N = x\end{aligned}$$

The RSA function

A modulus N and encryption exponent $e \in Z_{\varphi(N)}^*$ define the RSA function $f : Z_N^* \rightarrow Z_N^*$ via:

$$f(x) = x^e \pmod N$$

for all $x \in Z_N^*$.

A value $d \in Z_{\varphi(N)}^*$ satisfying $ed \pmod{\varphi(N)} = 1$ is called a decryption exponent.

Claim: The RSA function $f : Z_N^* \rightarrow Z_N^*$ is a permutation with inverse $f^{-1} : Z_N^* \rightarrow Z_N^*$ given by

$$f^{-1}(y) = y^d \pmod N$$

Proof: For all $x \in Z_N^*$, the prior claim says that we have

$$f^{-1}(f(x)) = (x^e)^d \pmod N = x .$$

Example

Let $N = 15$. So

$$Z_N^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

$$\varphi(N) = 8$$

$$Z_{\varphi(N)}^* = \{1, 3, 5, 7\}$$

Example

Let $N = 15$. So

$$\mathbb{Z}_N^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

$$\varphi(N) = 8$$

$$\mathbb{Z}_{\varphi(N)}^* = \{1, 3, 5, 7\}$$

Let $e = 3$ and $d = 3$. Then

$$ed \equiv 9 \equiv 1 \pmod{8}$$

Let

$$f(x) = x^3 \pmod{15}$$

$$g(y) = y^3 \pmod{15}$$

x	$f(x)$	$g(f(x))$
1	1	1
2	8	2
4	4	4
7	13	7
8	2	8
11	11	11
13	7	13
14	14	14

Trapdoor permutation

RSA is a trapdoor, one-way permutation:

- Easy to invert given trapdoor d
- Hard to invert given only N, e

The second is true, to best of our current knowledge, for appropriately-chosen parameters N, e, d .

The choice of parameters is done by an algorithm called an RSA generator.

RSA generators

An RSA generator with security parameter k is an algorithm \mathcal{K}_{rsa} that returns N, p, q, e, d satisfying

- p, q are distinct odd primes
- $N = pq$, and is called the (RSA) modulus
- $|N| = k$, meaning $2^{k-1} \leq N \leq 2^k$
- $e \in \mathbb{Z}_{\varphi(N)}^*$ is called the encryption exponent
- $d \in \mathbb{Z}_{\varphi(N)}^*$ is called the decryption exponent
- $ed \bmod \varphi(N) = 1$

A formula for Phi

Fact: Suppose $N = pq$ for distinct primes p and q . Then

$$\varphi(N) = (p - 1)(q - 1) .$$

Example: Let $N = 15 = 3 \cdot 5$. Then the Fact says that

$$\varphi(15) = (3 - 1)(5 - 1) = 8 .$$

As a check, $Z_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ indeed has size 8.

A more general formula for Phi

Fact: Suppose $N \geq 1$ factors as

$$N = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \dots \cdot p_n^{\alpha_n}$$

where $p_1 < p_2 < \dots < p_n$ are primes and $\alpha_1, \dots, \alpha_n \geq 1$ are integers.
Then

$$\varphi(N) = p_1^{\alpha_1-1}(p_1 - 1) \cdot p_2^{\alpha_2-1}(p_2 - 1) \cdot \dots \cdot p_n^{\alpha_n-1}(p_n - 1).$$

Note prior Fact is a special case of the above.

Example: Let $N = 45 = 3^2 \cdot 5^1$. Then the Fact says that

$$\varphi(45) = 3^1(3 - 1) \cdot 5^0(5 - 1) = 24$$

Given $\varphi(N)$ and $e \in Z_{\varphi(N)}^*$, we can compute $d \in Z_{\varphi(N)}^*$ satisfying $ed \bmod \varphi(N) = 1$ via

$$d \leftarrow \text{MOD-INV}(e, \varphi(N)).$$

We have algorithms to efficiently test whether a number is prime, and we know that a random number has a pretty good chance of being a prime.

We use these facts to build RSA generators.

Building RSA generators

Say we wish to have $e = 3$. (We will see that the smaller is e , the more efficient is encryption.) The generator \mathcal{K}_{rsa}^3 with (even) security parameter k is as follows:

repeat

$p, q \leftarrow^{\$} \{2^{k/2-1}, \dots, 2^{k/2} - 1\}; N \leftarrow pq; M \leftarrow (p-1)(q-1)$

until

$N \geq 2^{k-1}$ and p, q are prime and $\gcd(e, M) = 1$

$d \leftarrow \text{MOD-INV}(e, M)$

return N, p, q, e, d

One-wayness of RSA

The following should be hard:

Given: N, e, y where $y = f(x) = x^e \pmod N$

Find: x

Formalism picks x at random and generates N, e via an RSA generator.

One-wayness of RSA, formally

Let \mathcal{K}_{rsa} be a RSA generator and I an adversary.

Game $\text{OW}_{\mathcal{K}_{\text{rsa}}}$

procedure Initialize

$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$

$x \xleftarrow{\$} \mathbb{Z}_N^*$; $y \leftarrow x^e \pmod N$

return N, e, y

procedure Finalize(x')

return $(x = x')$

The ow-advantage of I is

$$\text{Adv}_{\mathcal{K}_{\text{rsa}}}^{\text{ow}}(I) = \Pr \left[\text{OW}_{\mathcal{K}_{\text{rsa}}}^I \Rightarrow \text{true} \right]$$

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

↑
EASY
Know d

because $x = y^d \bmod N$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$



EASY

Know d

because $x = y^d \bmod N$



EASY

Know $\varphi(N)$

because $d = \text{MOD-INV}(e, \varphi(N))$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$



EASY

Know d

because $x = y^d \bmod N$



EASY

Know $\varphi(N)$

because $d = \text{MOD-INV}(e, \varphi(N))$



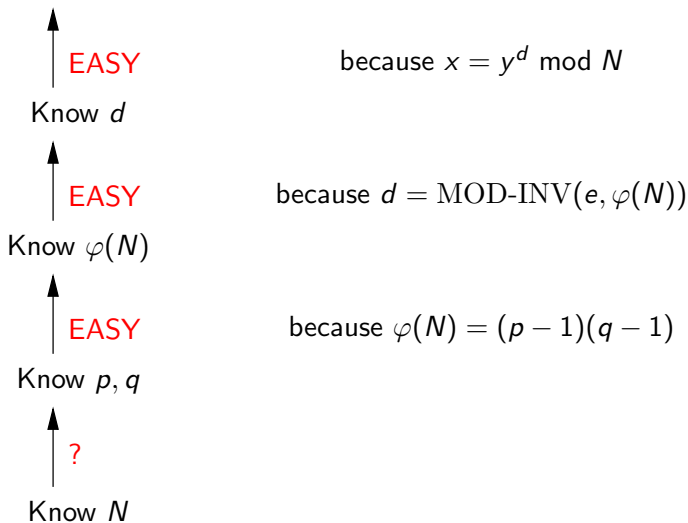
EASY

Know p, q

because $\varphi(N) = (p - 1)(q - 1)$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$



Factoring Problem

Given: N where $N = pq$ and p, q are prime

Find: p, q

If we can factor we can invert RSA. We do not know whether the converse is true, meaning whether or not one can invert RSA without factoring.

A factoring algorithm

Alg FACTOR(N) // $N = pq$ where p, q are primes

for $i = 2, \dots, \lceil \sqrt{N} \rceil$ do

 if $N \bmod i = 0$ then

$p \leftarrow i; q \leftarrow N/i$; return p, q

This algorithm works but takes time

$$\mathcal{O}(\sqrt{N}) = \mathcal{O}(e^{0.5 \ln N})$$

which is prohibitive.

Factoring algorithms

Algorithm	Time taken to factor N
Naive	$O(e^{0.5 \ln N})$
Quadratic Sieve (QS)	$O(e^{c(\ln N)^{1/2}(\ln \ln N)^{1/2}})$
Number Field Sieve (NFS)	$O(e^{1.92(\ln N)^{1/3}(\ln \ln N)^{2/3}})$

Factoring records

bit-length of number	When factored	Algorithm used
400	1993	QS
428	1994	QS
431	1996	NFS
465	1999	NFS
515	1999	NFS
576	2003	NFS
768	2009	NFS
795	2019	NFS
829	2020	NFS

We estimate that a 1024-bit RSA modulus provides 80 bits of security, meaning factoring it takes 2^{80} time.

Factorization of a 1024-bit modulus hasn't been done yet in public, but is within reach of large organizations. Longer moduli, like 2048 bits, have been recommended since around 2010.

Choices of encryption exponent

Common choices are $e = 3$, $e = 17$ and $e = 65,537$. Why these?

e	$\text{bin}(e)$
3	11
17	10001
65,537	1000000000000000001

Recall that the modular exponentiation algorithm computing $x \mapsto x^e \bmod N$ uses $c(b)$ modular multiplications per bit $b \in \{0, 1\}$ in the binary expansion $\text{bin}(e)$, where $c(0) = 1$ and $c(1) = 2$. So the fewer the number of 1s in $\text{bin}(e)$, the faster is the operation.

Low-exponent and other attacks

Further attacks on RSA include

- Coppersmith's attack
- Franklin-Reiter attack
- Håstad attack

These work for small encryption exponents but do not violate OW-security.

If RSA-based public-key encryption and digital signature schemes use RSA appropriately, these attacks do not threaten them, even if the encryption exponent is small.

Accordingly, in designing RSA-based public-key encryption and digital signature schemes, we seek proofs of security based (only) on the OW-security of RSA.

http://www.youtube.com/watch?v=wXB-V_Keiu8

The RSA function $f(x) = x^e \pmod N$ is a trapdoor one way permutation:

- Easy forward: given N, e, x it is easy to compute $f(x)$
- Easy back with trapdoor: Given N, d and $y = f(x)$ it is easy to compute $x = f^{-1}(y) = y^d \pmod N$
- Hard back without trapdoor: Given N, e and $y = f(x)$ it is hard to compute $x = f^{-1}(y)$

The quantum threat

On a quantum computer, Shor's algorithm can compute discrete logarithms and factor in polynomial time.

Efforts to build quantum computers are underway.

Efforts are underway to standardize public-key cryptography based on computational problems like finding short vectors in lattices for which there are currently no known efficient quantum algorithms.