Some mathematical foundations of Cryptography

PHAN Thi Ha Duong

Institute of Mathematics
Vietnam Academy of Science and Technology

IACR-SEAMS School
"Cryptography: Foundations and New Directions"
November, 2016

Introduction

- Problem: Alice and Bob want to share a commun secret key.
  But: Eve can observe every information that they exchange.

Introduction

- Problem: Alice and Bob want to share a commun secret key.
  But: Eve can observe every information that they exchange.
- Problem: Bob wants to send a ciphertext to Alice, using Alice's public key such that Alice can decrypt it to obtain the plaintext.

Introduction

- Problem: Alice and Bob want to share a commun secret key.
  But: Eve can observe every information that they exchange.
- Problem: Bob wants to send a ciphertext to Alice, using Alice's public key such that Alice can decrypt it to obtain the plaintext.
- Mathematical foundations in this talk:

Introduction

- Problem: Alice and Bob want to share a commun secret key. But: Eve can observe every information that they exchange.
- Problem: Bob wants to send a ciphertext to Alice, using Alice's public key such that Alice can decrypt it to obtain the plaintext.
- Mathematical foundations in this talk:
- Discrete Logarithms.

Introduction

- Problem: Alice and Bob want to share a commun secret key.
  But: Eve can observe every information that they exchange.
- Problem: Bob wants to send a ciphertext to Alice, using Alice's public key such that Alice can decrypt it to obtain the plaintext.
- Mathematical foundations in this talk:
- Discrete Logarithms.
- Integer Factorizations.

Introduction

- Problem: Alice and Bob want to share a commun secret key. But: Eve can observe every information that they exchange.
- Problem: Bob wants to send a ciphertext to Alice, using Alice's public key such that Alice can decrypt it to obtain the plaintext.
- Mathematical foundations in this talk:
- Discrete Logarithms.
- Integer Factorizations.
- Propeties:
  - easy to compute on every input
  - hard to invert the image of a random input

  easy: polynomial time

  hard: exponential time.

Introduction

- Some notions on Complexity

Introduction

- Some notions on Complexity
- Number-Theoretic Algorithms

Introduction

- Some notions on Complexity
- Number-Theoretic Algorithms
- Discrete Logarithms and Public Key Problem.
  - Diffie- Hellman Key Exchange
  - The Elgamal Public Key Cryptosystem
  - Babystep - Giantstep Algorithm
  - The Pohlig- Hellman Algorithm
  - The index calculus method

Introduction

- Some notions on Complexity
- Number-Theoretic Algorithms
- Discrete Logarithms and Public Key Problem.
  - Diffie- Hellman Key Exchange
  - The Elgamal Public Key Cryptosystem
  - Babystep - Giantstep Algorithm
  - The Pohlig- Hellman Algorithm
  - The index calculus method
- RSA and Integer Factorization
  - Pollard's p-1 Factorization
  - Factorization via Difference of Squares
  - B- smooth number

Asymptotic notations

The complexity of an algorithm is represented by a function $f(N)$ where N is the size of the input.

Definition.

Let $f(X)$ and $g(X)$ be functions of X whose values are positive. Then we define the following notions

- $f(X) = O(g(X))$ *if there exists constants c and X_0 such that* $f(X) \leq cg(X)$ *for all* $X \geq X_0$.
- $f(X) = \Omega(g(X))$ *if there exists constants c and X_0 such that* $f(X) \geq cg(X)$ *for all* $X \geq X_0$.
- $f(X) = \Theta(g(X))$ *if* $f(X) = O(g(X))$ *and* $f(X) = \Omega(g(X))$.
- $f(X) = o(g(X))$ *if for all constant c, there exists a constant X_o such that* $f(X) < cg(X)$ *for all* $X \geq X_0$.
- $f(X) = \omega(g(X))$ *if for all constant c, there exists a constant X_o such that* $f(X) > cg(X)$ *for all* $X \geq X_0$.

Asymtotically,

$$f(X) = O(g(X)) \Leftrightarrow f(X) \leq g(X)$$

$$f(X) = \Omega(g(X)) \Leftrightarrow f(X) \geq g(X)$$

$$f(X) = \Theta(g(X)) \Leftrightarrow f(X) \sim g(X)$$

$$f(X) = o(g(X)) \Leftrightarrow f(X) < g(X)$$

$$f(X) = \omega(g(x)) \Leftrightarrow f(X) > g(X).$$

Polynomial, exponential and subexponential

For the input being a large number X (the size is $\log X$), the complexity $f(X)$ is considered as a function of $\log X$.

Definition.

The complexity grows polynomially if
$\exists k, l$: $f(X) = O((\log X)^k) \quad \& \quad f(X) = \Omega((\log X)^l)$.
The complexity grows exponentially if
$\exists k, l$: $f(X) = O((X)^k) \quad \& \quad f(X) = \Omega((X)^l)$.
The complexity is subexponential if
$\forall k, l$: $f(X) = O((X)^k) \quad \& \quad f(X) = \Omega((\log X)^l)$.

Example

$f_1(X) = (\log X)^3 \log\log X \sqrt{\log X}, \quad f_2(X) = \frac{1}{3}X, \quad f_3(X) = \sqrt{X}$
$f_4(X) = e^{\sqrt{(lnX)(lnlnX)}}$.

Number-Theoretic Algorithms

- The Euclidean Algorithm
- Prime number and Factorization
- Powers and primitive roots in finite fields
- The Chinese reminder Theorem

The Euclidean Algorithm

Problem.

Let a and b be positive integers with $a \geq b$. Find the greatest comment divisor of a and b ($gcd(a, b)$).

Algorithm.

1. *Let* $r_0 := a, r_1 := b, i := 1$;
2. *Devide* r_{i-1} *by* r_i: $r_{i-1} = r_i q_i + r_{i+1}$ with $0 \leq r_{i+1} < r_i$.
3. *If* $r_{i+1} = 0$ *then* $gcd(a, b) := r_i$
4. *Otherwise,* $i := i + 1$; *go to Step 2.*

Complexity $O(\log b)$: linear time.

The Extended Euclidean Algorithm

Theorem.

Let a and b be positive integers with $a \geq b$. Then the equation $au + bv = gcd(a, b)$ always has a solution in integer u and v.

Using the Euclidean Algorithm, then we can find u and v as functions of q_i.

Complexity $O(\log b)$.

Proposition.

Let a be an integer, then there exists integer b such that $a.b \equiv 1 (\mod m)$ if and only if $gcd(a, m) = 1$. If such an integer b exists, we say that b is the (multiplicative) inverse of a modulo m.
Moreover b can be found in $O(\log m)$.
In particular, if p is prime, then the inverse of a in $\mathbb{F}_p^*$ exists always, and denoted by a^{-1}.

Prime number and Factorization

Theorem.

(The Fundamental Theorem of Arithmetic)
Let $a \geq 2$ be an integer. Then a can be factored as a product of prime numbers in a unique way

$$a = p_1^{e_1}.p_2^{e_2}.\ldots.p_r^{e_r}.$$

The number e_i is called the order of p_i in a, denoted by $ord_{p_i}(a)$.

Powers and primitive roots in finite fields

Theorem.

(Fermat's Little Theorem) Let p be a prime number and let a be any integer. Then $a^{p-1} \equiv 1 \pmod p$ if p does not devide a.

Definition.

The order of a modulo p is the smallest power of a that are congruent to 1: $a^k \equiv 1 \pmod p$.

Proposition.

Let p be a prime and let a be an integer not divisible by p. Then the order of a divides $p - 1$.

Primitive Root Theorem

Theorem.

(Primitive Root Theorem)
Let p be a prime number. Then there exists an element $a \in \mathbb{F}_p^*$ whose powers give every elements of $\mathbb{F}_p^*$, i.e.

$$\mathbb{F}_p^* = \{1, g, g^2, \ldots, g^{p-2}\}.$$

Elements with this property are called primitive roots of $\mathbb{F}_p$ or generator of $\mathbb{F}_p^*$. They are of order $p-1$.

Example.

The field $\mathbb{F}_{11}^*$ has 2 as a primitive root, since in $\mathbb{F}_{11}^*$:

$$2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 = 5, 2^5 = 10, 2^6 = 9, 2^7 = 7, 2^8 = 3, 2^9 = 6$$

and 3 is not a primitive in $\mathbb{F}_{11}^*$ because $3^5 = 1$ (note that 10 is divisible by 5).

The Chinese reminder Theorem

Theorem.

Let $n = n_1 n_2 \dots n_k$, where n_i are pairwise relatively prime. Then the map

$$f : \mathbb{Z}_n \to \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \dots \times \mathbb{Z}_{n_k}$$

$$a \to (a_1, a_2, \dots, a_k), \ \textit{where } a_i = a \textit{ mod } n_i$$

is a bijection. And

$(a + b) \textit{ mod } n = ((a_1 + b_1) \textit{ mod } n_1, \dots, (a_k + b_k) \textit{ mod } n_k)$

$(a - b) \textit{ mod } n = ((a_1 - b_1) \textit{ mod } n_1, \dots, (a_k - b_k) \textit{ mod } n_k)$

$(ab) \textit{ mod } n = ((a_1 b_1) \textit{ mod } n_1, \dots, (a_k b_k) \textit{ mod } n_k)$

Proof

From $(a_1, a_2, \dots a_k)$, how to find a ?

Let $m_i = n/n_i$. Compute $c_i = m_i(m_i^{-1} \bmod n_i)$.

Then $f(c_i) = (0, \dots, 0, 1, 0 \dots, 0)$

Take $a = c_1 a_1 + c_2 a_2 + \dots + c_k a_k \pmod{n}$, then $a \equiv a_i \pmod{n_i}$.

The Chinese reminder Theorem

Corollary.

If $n = n_1 n_2 \ldots n_k$, where the n_i are pairwise relatively prime, then for any integer $a_1, a_2, \ldots, a_k$, the system of equations $x \equiv a_i (mod\ n_i)$ for $i = 1, 2, \ldots, k$, has a unique solution modulo n for x.

Corollary.

If $n = n_1 n_2 \ldots n_k$, where the n_i are pairwise relatively prime, then for all integer x and a, $x \equiv a \pmod{n_i}$ for $i = 1, 2, \ldots, k$, if and only if $x \equiv a \pmod{n}$.

The complexity to solve these equations is $O(\log n)$.

Discrete Logarithms and Diffie- Hellman

- Diffie- Hellman Key Exchange
- Diffie-Hellman Problem
- The Discrete Logarithm Problem
- The Elgamal Public Key Cryptosystem
- Babystep - Giantstep Algorithm
- The Pohlig- Hellman Algorithm
- The index calculus method

Diffie- Hellman Key Exchange

Problem.

Alice and Bob want to share a secret key.
But: Eve can observe every information that they exchange.

Algorithm.

Public Parameter creation *A trusted party gives:*
p: a large prime number and g: a large prime order in $\mathbb{F}_p^*$.
Private computations
Alice: chose a secret interger a. Compute $A = g^a \mod p$.
Bob: chose a secret interger b. Compute $B = g^b \mod p$.
Public exchange of value
Alice sends A to Bob. Bob sends B to Alice.
Further private computations
Alice: Compute $K_A = B^a \mod p$. *Bob: Compute* $K_B = A^b \mod p$.
Property: $K = K_A = K_B$*: Alice and Bob share a Secret Key* K.

The Diffie-Hellman Problem

Eve:

- Know p, g, g^a and g^b.
- Want to know g^{ab}.

Definition.

Let p be a prime number and g an integer. The Diffie- Hellman Problem (DHP) is the problem of computing the value $g^{ab} \pmod p$ from the known values of $g^a \pmod p$ and $g^b \pmod p$.

The Discrete Logarithm Problem

Definition.

Let g be a primitive root for $\mathbb{F}_p$, and let h be a nonzero element of $\mathbb{F}_p$. The Discrete Logarithm Problem (DLP) is the problem of finding an exponent x such that $g^x \equiv h \pmod{p}$.
The number x is called the discrete logarithm of h to the base g, denoted by $log_g(h)$.

- In $\mathbb{F}_p$, if $g^x = h$ then $g^{x+k(p-1)} = h$.
So $log_g : \mathbb{F}_p^* \to \mathbb{Z}/(p-1)\mathbb{Z}$.
- In general, if g is not a primitive root of $\mathbb{F}_p^*$, one can also define the DLP: for any $g \in \mathbb{F}_p^*$ and any $h \in \mathbb{F}_p^*$, find x such that $g^x \equiv h \pmod{p}$.

The Discrete Logarithm Problem

Definition.

Let G be a group with operation $\star$. The Discrete Logarithm Problem (DLP) for G is to determine, for any two given elements g and h in G, an integer x satisfying $\underbrace{g \star g \ldots \star g}_{x} = h$.

DHP vs DLP

- If one can solve the DLP then one can solve the DHP.
  If one can find a such that $g^a \equiv A \pmod{p}$, than one can compute $g^{ab} = B^a$ (with B being g^b) and solving DHP.

The Discrete Logarithm Problem

Definition.

Let G be a group with operation $\star$. The Discrete Logarithm Problem (DLP) for G is to determine, for any two given elements g and h in G, an integer x satifying $\underbrace{g \star g \ldots \star g}_{x} = h$.

DHP vs DLP

- If one can solve the DLP then one can solve the DHP.
  If one can find a such that $g^a \equiv A \pmod{p}$, than one can compute $g^{ab} = B^a$ (with B being g^b) and solving DHP.
- Open question: If one can solve the DHP then one can solve the DLP or not?

The Elgamal Public Key Cryptosystem

Problem.

Bob wants to send a ciphertext to Alice, using Alice's public key.

Algorithm.

Public Parameter creation A trusted party gives:
p: a large prime number and g: a large prime order in $\mathbb{F}_p^*$.
Key creation Alice:
Choose a private key $1 \leq a \leq p-1$. Compute $A = g^a \mod p$).
Publish the public key A.
Encryption Bob:
Choose plaintext m. Choose a random element k.
Compute $c_1 = g^k \mod p$ and $c_2 = mA^k \mod p$.
Send ciphertext (c_1, c_2) to Alice.
Decryption Alice:
Compute $m' = c_1^{-a} c_2 \mod p$. This is equal to m.

Corectness and Complexity

Corectness

$m' \equiv c_1^{-a} c_2 \equiv (g^k)^{-a} m A^k \equiv g^{-ak} m g^{ak} \equiv m \pmod{p}$.

- Alice compute c_1^{-a} or c_1^{p-1-a} by using fast powering.

Complexity

Every step in the system is computed in linear time.

Attack:

- Eve knows: g, p and A.
- If Eve knows DLP, she can find a, and then compute m' as Alice.

DHP and Elgamal PKC

Proposition.

If Eve has access to an oracle that decryps arbitraty Elgamal ciphertext encryptes using arbitrary Elgamal public keys, then she can use the oracle to solve the Diffie- Hellman problem.
Conversly, if Eve can solve the DHP, then she can break the Elgamal PKC.

Proof.

Suppose that Eve can consult an Elgama oracle.
To solve DHP: Eve knows $A = g^a$ and $B = g^b$ (but not a and b), and Eve must to compute g^{ab}
Now, Eve choose: public key A, $c_1 = B$ and an arbitrary c_2. Send to the oracle.
The oracle return $m = c_1^{-a} c_2 = B^{-a} c_2 = (g^{ab})^{-1} c_2$
Then $g^{ab} = m^{-1} c_2$.

Complexity of DLP

- If Eve can solve DLP, she can solve DHP and Elgamal PKC.

Definition.

Let G be a group with operation $\star$. The Discrete Logarithm Problem (DLP) for G is to determine, for any two given element g and h in G, an integer x satifying $g \star g \ldots \star g \equiv h$.

Complexity of DLP

- If Eve can solve DLP, she can solve DHP and Elgamal PKC.

Definition.

Let G be a group with operation $\star$. The Discrete Logarithm Problem (DLP) for G is to determine, for any two given element g and h in G, an integer x satifying $g \star g \ldots \star g \equiv h$.

- If G is the additive group $\mathbb{F}_p$, then DLP is to compute x such that $x.g \equiv h \pmod{p}$. This is in linear time.
  Proof. By extended Euclidean algorithm, in linear time, compute $g^{-1} \pmod{p}$, and setting $x = g^{-1}h \pmod{p}$.

Complexity of DLP

- If Eve can solve DLP, she can solve DHP and Elgamal PKC.

Definition.

Let G be a group with operation $\star$. The Discrete Logarithm Problem (DLP) for G is to determine, for any two given element g and h in G, an integer x satifying $g \star g \ldots \star g \equiv h$.

- If G is the additive group $\mathbb{F}_p$, then DLP is to compute x such that $x.g \equiv h \pmod{p}$. This is in linear time.
  Proof. By extended Euclidean algorithm, in linear time, compute $g^{-1} \pmod{p}$, and setting $x = g^{-1}h \pmod{p}$.
- If G is a group of elliptic curves: the best know algorithm for DLP is $O(\sqrt{N})$ (so exponential).

Complexity of DLP

- If Eve can solve DLP, she can solve DHP and Elgamal PKC.

Definition.
Let G be a group with operation $\star$. The Discrete Logarithm Problem (DLP) for G is to determine, for any two given element g and h in G, an integer x satifying $g \star g \ldots \star g \equiv h$.

- If G is the additive group $\mathbb{F}_p$, then DLP is to compute x such that $x.g \equiv h \pmod{p}$. This is in linear time.
  Proof. By extended Euclidean algorithm, in linear time, compute $g^{-1} \pmod{p}$, and setting $x = g^{-1}h \pmod{p}$.
- If G is a group of elliptic curves: the best know algorithm for DLP is $O(\sqrt{N})$ (so exponential).
- If G is the multiplicative group $\mathbb{F}_p^*$, DLP is subexponential: Algorithms ?

Babystep - Giantstep Algorithm

Proposition.

Let g be a group and $g \in G$ of order $N \geq 2$. The following algorithm solve the DLP $g^x = h$ in $O(\sqrt{N} logN)$ steps using $O(\sqrt{N})$ storage.
(1) Let $n = 1 + \lfloor \sqrt{N} \rfloor$.
(2) Create two lists:

$$List1 : e, g, g^2, \ldots, g^n,$$

$$List2 : h, hg^{-n}, hg^{-2n}, hg^{-3n}, \ldots, hg^{-n^2}.$$

(3) Find i and j such that $g^i = hg^{-jn}$ ($\Leftrightarrow g^{i+jn} = h$).
(4) Then $x = i + jn$ is a solution.

Babystep - Giantstep Algorithm

Corectness

If DLP has a solution x, then write $x : qn + r, 0 \leq r < n$.
$1 \leq x < N$ then $q = \frac{x-r}{n} < \frac{N}{n} < n$ since $n > \sqrt{N}$.
Then $g^x = h \Leftrightarrow g^r = hg^{-qn}$ with $0 \leq r < n$ and $0 \leq q < n$, then

$$g^r \in List1 \text{ and } hg^{-qn} \in List2.$$

Complexity

(1) and (4): $O(1)$
(2) Compute: $u = g^{-n}$.
Compute List 1 in $O(n)$ multiplications.
Compute List 2 in $O(n)$ multiplications.
(3) Finding a match by using sorting and searching: $O(nlogn)$.
Total time: $O(nlogn) = O(\sqrt{N}logN)$ time, using $O(\sqrt{N})$ space to store List 1 and List 2.

The Pohlig- Hellman Algorithm

Theorem.

Let G be a group and $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$ (factorization of N). If the DLP $g^q = h$ for g of order q can be solved in $T(q)$ time, then the DLP for g of order N can be solved in time

$$O(\sum_{i=1}^{t} e_i T(q) + logN).$$

Remark.

- The $T(q)$ can be $O(\sqrt{q})$ then $T(N) = O(\sum_{i=1}^{t} e_i \sqrt{q_i} + logN)$.

The Pohlig- Hellman Algorithm

Theorem.

Let G be a group and $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$ (factorization of N). If the DLP $g^q = h$ for g of order q can be solved in $T(q)$ time, then the DLP for g of order N can be solved in time

$$O(\sum_{i=1}^{t} e_i T(q) + logN).$$

Remark.

- The $T(q)$ can be $O(\sqrt{q})$ then $T(N) = O(\sum_{i=1}^{t} e_i\sqrt{q_i} + logN)$.
- If all q_i are small then $T(N)$ is small.

The Pohlig- Hellman Algorithm

Theorem.

Let G be a group and $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$ (factorization of N). If the DLP $g^q = h$ for g of order q can be solved in $T(q)$ time, then the DLP for g of order N can be solved in time

$$O(\sum_{i=1}^{t} e_i T(q) + logN).$$

Remark.

- The $T(q)$ can be $O(\sqrt{q})$ then $T(N) = O(\sum_{i=1}^{t} e_i \sqrt{q_i} + logN)$.
- If all q_i are small then $T(N)$ is small.
- To avoid the attack, some of q_i must be large, ie. the base g must be a large prime order.

Proof of the Pohlig- Hellman Theorem

Proof.

- For $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$ then $T(N) = O(\sum_{i=1}^{t} T(q_i^{e_i}) + logN)$. Using Chinese remainder theorem.

Proof of the Pohlig- Hellman Theorem

Proof.

- For $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$ then $T(N) = O(\sum_{i=1}^{t} T(q_i^{e_i}) + logN)$. Using Chinese remainder theorem.
- For $N = q^e$ then $T(q^e) = O(eT(q))$.

Proof of the Pohlig- Hellman Theorem. Part 1

- Let $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$.
  Let $g_i = g^{N/q_i^{e_i}}$ and $h_i = h^{N/q_i^{e_i}}$. Then g_i is of order $q_i^{e_i}$.
  Find the solution y_i of the DLP $g_i^y = h_i$ in $T(q_i^{e_i})$ time.

Proof of the Pohlig- Hellman Theorem. Part 1

- Let $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$.
  Let $g_i = g^{N/q_i^{e_i}}$ and $h_i = h^{N/q_i^{e_i}}$. Then g_i is of order $q_i^{e_i}$.
  Find the solution y_i of the DLP $g_i^y = h_i$ in $T(q_i^{e_i})$ time.
- Use the Chinese reminder Theorem in $O(logN)$ time to solve $x \equiv y_1 \pmod{q_1^{e_1}}$, $x \equiv y_2 \pmod{q_2^{e_2}}$, ..., $x \equiv y_t \pmod{q_t^{e_t}}$.

Proof of the Pohlig- Hellman Theorem. Part 1

- Let $N = q_1^{e_1}.q_2^{e_2}.\ldots.q_t^{e_t}$.
  Let $g_i = g^{N/q_i^{e_i}}$ and $h_i = h^{N/q_i^{e_i}}$. Then g_i is of order $q_i^{e_i}$.
  Find the solution y_i of the DLP $g_i^y = h_i$ in $T(q_i^{e_i})$ time.
- Use the Chinese reminder Theorem in $O(logN)$ time to solve
  $x \equiv y_1 \pmod{q_1^{e_1}}$, $x \equiv y_2 \pmod{q_2^{e_2}}$, ..., $x \equiv y_t \pmod{q_t^{e_t}}$.
- For each i, $x = y_i + q_i^{e_i} z_i$ for some z_i.
  $\Rightarrow (g^x)^{N/q_i^{e_i}} = (g^{y_i + q_i^{e_i} z_i})^{N/q_i^{e_i}} = (g^{N/q_i^{e_i}})^{y_i}.g^{Nz_i} = g_i^{y_i} = h_i = h^{N/q_i^{e_i}}$.
  Taking discrete logarithms to the base g: $\frac{N}{q_i^{e_i}}.x \equiv \frac{N}{q_i^{e_i}}.log_g h \pmod{N}$.

Proof of the Pohlig- Hellman Theorem. Part 1

- Let $N = q_1^{e_1}.q_2^{e_2}.\dots.q_t^{e_t}$.
  Let $g_i = g^{N/q_i^{e_i}}$ and $h_i = h^{N/q_i^{e_i}}$. Then g_i is of order $q_i^{e_i}$.
  Find the solution y_i of the DLP $g_i^y = h_i$ in $T(q_i^{e_i})$ time.
- Use the Chinese reminder Theorem in $O(logN)$ time to solve
  $x \equiv y_1 \pmod{q_1^{e_1}}$, $x \equiv y_2 \pmod{q_2^{e_2}}$, ..., $x \equiv y_t \pmod{q_t^{e_t}}$.
- For each i, $x = y_i + q_i^{e_i} z_i$ for some z_i.
  $\Rightarrow (g^x)^{N/q_i^{e_i}} = (g^{y_i + q_i^{e_i} z_i})^{N/q_i^{e_i}} = (g^{N/q_i^{e_i}})^{y_i}.g^{Nz_i} = g_i^{y_i} = h_i = h^{N/q_i^{e_i}}$.
  Taking discrete logarithms to the base g: $\frac{N}{q_i^{e_i}}.x \equiv \frac{N}{q_i^{e_i}}.log_g h \pmod{N}$.
- $\frac{N}{q_1^{e_1}}, \frac{N}{q_2^{e_2}}, \dots, \frac{N}{q_t^{e_t}}$ have no common factor, then $\exists c_1, c_2, \dots, c_t$:
  $c_1.\frac{N}{q_1^{e_1}}, +c_2.\frac{N}{q_2^{e_2}}, + \dots + c_t.\frac{N}{q_t^{e_t}} = 1$
  $\Rightarrow \sum_{i=1}^{t} c_i \frac{N}{q_i^{e_i}}.x \equiv \sum_{i=1}^{t} c_i \frac{N}{q_i^{e_i}} log_g h \pmod{N}$.
  $\Rightarrow x = log_g h \pmod{N} \Rightarrow g^x \equiv h \pmod{p}$.

Proof of the Pohlig- Hellman Theorem. Part 2

- If the order of g is q^e. Finding x: $g^x = h$ is in time $O(eT(q))$?

Proof of the Pohlig- Hellman Theorem. Part 2

- If the order of g is q^e. Finding x: $g^x = h$ is in time $O(eT(q))$?
- Write $x = x_0 + x_1 q + \dots x_{e-1} q^{e-1}$, with $0 \leq x_i < q$.

Proof of the Pohlig- Hellman Theorem. Part 2

- If the order of g is q^e. Finding x: $g^x = h$ is in time $O(eT(q))$?
- Write $x = x_0 + x_1 q + \dots x_{e-1} q^{e-1}$, with $0 \leq x_i < q$.
- Note that $g^{q^{e-1}}$ is of order q. Find x_0:
  $h^{q^{e-1}} = (g^x)^{q^{e-1}} = (g^{x_0 + x_1 q + \dots x_{e-1} q^{e-1}})^{q^{e-1}} =$
  $g^{x_0 q^{e-1}}.(g^{q^e})^{x_1 + x_2 q + \dots x_{e-1} q^{e-2}} = (g^{q^{e-1}})^{x_0}$.
  Since $g^{q^{e-1}}$ is of order q, then finding x_0 such that $(g^{q^{e-1}})^{x_0} = h^{q^{e-1}}$ is in $T(q)$ time.

Proof of the Pohlig- Hellman Theorem. Part 2

- If the order of g is q^e. Finding x: $g^x = h$ is in time $O(eT(q))$?
- Write $x = x_0 + x_1 q + \ldots x_{e-1} q^{e-1}$, with $0 \leq x_i < q$.
- Note that $g^{q^{e-1}}$ is of order q. Find x_0:
  $h^{q^{e-1}} = (g^x)^{q^{e-1}} = (g^{x_0 + x_1 q + \ldots x_{e-1} q^{e-1}})^{q^{e-1}} =$
  $g^{x_0 q^{e-1}} . (g^{q^e})^{x_1 + x_2 q + \ldots x_{e-1} q^{e-2}} = (g^{q^{e-1}})^{x_0}$.
  Since $g^{q^{e-1}}$ is of order q, then finding x_0 such that $(g^{q^{e-1}})^{x_0} = h^{q^{e-1}}$ is in $T(q)$ time.
- Finding x_1: $h^{q^{e-2}} = (g^x)^{q^{e-2}} = (g^{x_0 + x_1 q + \ldots x_{e-1} q^{e-1}})^{q^{e-2}} =$
  $g^{x_0 q^{e-2}} . g^{x_1 q^{e-1}} . (g^{q^e})^{x_2 + x_3 q + \ldots x_{e-1} q^{e-3}} = g^{x_0 q^{e-2}} . (g^{q^{e-1}})^{x_1}$.
  $\Rightarrow (g^{q^{e-1}})^{x_1} = (h . g^{-x_0})^{q^{e-2}}$.
  Finding x_1 is in $T(q)$ time.

Proof of the Pohlig- Hellman Theorem. Part 2

- If the order of g is q^e. Finding x: $g^x = h$ is in time $O(eT(q))$?
- Write $x = x_0 + x_1 q + \dots x_{e-1} q^{e-1}$, with $0 \leq x_i < q$.
- Note that $g^{q^{e-1}}$ is of order q. Find x_0:
  $h^{q^{e-1}} = (g^x)^{q^{e-1}} = (g^{x_0 + x_1 q + \dots x_{e-1} q^{e-1}})^{q^{e-1}} =$
  $g^{x_0 q^{e-1}}.(g^{q^e})^{x_1 + x_2 q + \dots x_{e-1} q^{e-2}} = (g^{q^{e-1}})^{x_0}$.
  Since $g^{q^{e-1}}$ is of order q, then finding x_0 such that $(g^{q^{e-1}})^{x_0} = h^{q^{e-1}}$ is in $T(q)$ time.
- Finding x_1: $h^{q^{e-2}} = (g^x)^{q^{e-2}} = (g^{x_0 + x_1 q + \dots x_{e-1} q^{e-1}})^{q^{e-2}} =$
  $g^{x_0 q^{e-2}}.g^{x_1 q^{e-1}}.(g^{q^e})^{x_2 + x_3 q + \dots x_{e-1} q^{e-3}} = g^{x_0 q^{e-2}}.(g^{q^{e-1}})^{x_1}$.
  $\Rightarrow (g^{q^{e-1}})^{x_1} = (h.g^{-x_0})^{q^{e-2}}$.
  Finding x_1 is in $T(q)$ time.
- and so on for $x_2, \dots, x_{e-1}$.

Proof of the Pohlig- Hellman Theorem. Part 2

- If the order of g is q^e. Finding x: $g^x = h$ is in time $O(eT(q))$?
- Write $x = x_0 + x_1 q + \dots x_{e-1} q^{e-1}$, with $0 \leq x_i < q$.
- Note that $g^{q^{e-1}}$ is of order q. Find x_0:
  $h^{q^{e-1}} = (g^x)^{q^{e-1}} = (g^{x_0 + x_1 q + \dots x_{e-1} q^{e-1}})^{q^{e-1}} =$
  $g^{x_0 q^{e-1}}.(g^{q^e})^{x_1 + x_2 q + \dots x_{e-1} q^{e-2}} = (g^{q^{e-1}})^{x_0}$.
  Since $g^{q^{e-1}}$ is of order q, then finding x_0 such that $(g^{q^{e-1}})^{x_0} = h^{q^{e-1}}$ is in $T(q)$ time.
- Finding x_1: $h^{q^{e-2}} = (g^x)^{q^{e-2}} = (g^{x_0 + x_1 q + \dots x_{e-1} q^{e-1}})^{q^{e-2}} =$
  $g^{x_0 q^{e-2}}.g^{x_1 q^{e-1}}.(g^{q^e})^{x_2 + x_3 q + \dots x_{e-1} q^{e-3}} = g^{x_0 q^{e-2}}.(g^{q^{e-1}})^{x_1}$.
  $\Rightarrow (g^{q^{e-1}})^{x_1} = (h.g^{-x_0})^{q^{e-2}}$.
  Finding x_1 is in $T(q)$ time.
- and so on for $x_2, \dots, x_{e-1}$.
- The total time is then $eT(q)$.

Example of the Pohlig- Hellman Algorithm

Problem.

Problem: Find x *such that* $23^x = 9689$ *in* $\mathbb{F}_{11251}$.

Algorithm.

- $11250 = 2.3^2.5^4$, *and* 23 *is primitive (of order* 11250*) in* $\mathbb{F}_{11251}$.
  $p = 11251, g = 23, h = 9689, N = p - 1 = 2.3^2.5^4$
-

q	e	$g^{(N/q^e)}$	$h^{(N/q^e)}$	Solve $(g^{(N/q^e)})^x = h^{(N/q^e)}$
2	1	11250	11250	1
3	2	5029	10724	4
5	4	5448	6909	511

- *Chinese remainder theorem, solve:* $x \equiv 1 \pmod 2$, $x \equiv 4 \pmod{3^2}$, $x \equiv 511 \pmod{5^4}$. *Then* $x = 4261$.
  Then $23^{4261} = 9689$ *in* $\mathbb{F}_{11251}$.

For example, solve: $5448^x = 6909 \pmod{11251}$.

$$x = x_0 + x_1.5 + x_2.5^2 + x_3.5^3.$$

- Finding x_0: $(5448^{5^3})^{x_0} = 6909^{5^3}$, $\Leftrightarrow 11089^{x_0} = 11089 \Rightarrow x_0 = 1$.

For example, solve: $5448^x = 6909 \pmod{11251}$.

$$x = x_0 + x_1.5 + x_2.5^2 + x_3.5^3.$$

- Finding x_0: $(5448^{5^3})^{x_0} = 6909^{5^3}$, $\Leftrightarrow 11089^{x_0} = 11089 \Rightarrow x_0 = 1$.
- Finding x_1: $(5448^{5^3})^{x_1} = (6909.5448^{-x_0})^{5^2} = (6909.5448^{-1})^{5^2}$ $\Leftrightarrow 11089^{x_1} = 3742 \Rightarrow x_1 = 2$.

For example, solve: $5448^x = 6909 \pmod{11251}$.

$$x = x_0 + x_1.5 + x_2.5^2 + x_3.5^3.$$

- Finding x_0: $(5448^{5^3})^{x_0} = 6909^{5^3}$, $\Leftrightarrow 11089^{x_0} = 11089 \Rightarrow x_0 = 1$.
- Finding x_1: $(5448^{5^3})^{x_1} = (6909.5448^{-x_0})^{5^2} = (6909.5448^{-1})^{5^2}$ $\Leftrightarrow 11089^{x_1} = 3742 \Rightarrow x_1 = 2$.
- Finding x_2: $(5448^{5^3})^{x_2} = (6909.5448^{-x_0 - x_1.5})^5 = (6909.5448^{-11})^5$ $\Leftrightarrow 11089^{x_2} = 1 \Rightarrow x_2 = 0$.

For example, solve: $5448^x = 6909 \pmod{11251}$.

$$x = x_0 + x_1.5 + x_2.5^2 + x_3.5^3.$$

- Finding x_0: $(5448^{5^3})^{x_0} = 6909^{5^3}$, $\Leftrightarrow 11089^{x_0} = 11089 \Rightarrow x_0 = 1$.
- Finding x_1: $(5448^{5^3})^{x_1} = (6909.5448^{-x_0})^{5^2} = (6909.5448^{-1})^{5^2}$
  $\Leftrightarrow 11089^{x_1} = 3742 \Rightarrow x_1 = 2$.
- Finding x_2: $(5448^{5^3})^{x_2} = (6909.5448^{-x_0-x_1.5})^5 = (6909.5448^{-11})^5$
  $\Leftrightarrow 11089^{x_2} = 1 \Rightarrow x_2 = 0$.
- Finding x_3: $(5448^{5^3})^{x_3} = 6909.5448^{-x_0-x_1.5-x_2.5^2} = 6909.5448^{-11}$
  $\Leftrightarrow 11089^{x_3} = 6320 \Rightarrow x_3 = 4$.

For example, solve: $5448^x = 6909 \pmod{11251}$.

$$x = x_0 + x_1.5 + x_2.5^2 + x_3.5^3.$$

- Finding x_0: $(5448^{5^3})^{x_0} = 6909^{5^3}$, $\Leftrightarrow 11089^{x_0} = 11089 \Rightarrow x_0 = 1$.
- Finding x_1: $(5448^{5^3})^{x_1} = (6909.5448^{-x_0})^{5^2} = (6909.5448^{-1})^{5^2}$ $\Leftrightarrow 11089^{x_1} = 3742 \Rightarrow x_1 = 2$.
- Finding x_2: $(5448^{5^3})^{x_2} = (6909.5448^{-x_0-x_1.5})^5 = (6909.5448^{-11})^5$ $\Leftrightarrow 11089^{x_2} = 1 \Rightarrow x_2 = 0$.
- Finding x_3: $(5448^{5^3})^{x_3} = 6909.5448^{-x_0-x_1.5-x_2.5^2} = 6909.5448^{-11}$ $\Leftrightarrow 11089^{x_3} = 6320 \Rightarrow x_3 = 4$.
- $x = 1 + 2.5 + 4.5^3 = 511$.

The index calculus method. Smooth numbers

Definition.

An integer n is called B-smooth if all of its prime factors are less than or equal to B.

Definition.

The function $\pi(B)$ counts prime numbers that are smaller than B.

Example $B = 5$, $\pi(5) = 3$.
$5 - smooths : 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30, 32, 36, \ldots$
Not $5 - smooths$:
$7, 11, 13, 14, 17, 19, 21, 23, 26, 28, 29, 31, 33, 34, 35, 37, \ldots$

The index calculus method

Problem.

Let g be a primitive root of $\mathbb{F}_p$, find x st: $g^x \equiv h \pmod p$.

Algorithm.

- *Solve problem $g^x \equiv \ell \pmod p$ for all prime $\ell \leq B$.*
- *Look at $h.g^{-k} \mod p$ for $k = 1, 2, \ldots$ until a value k such that $h.g^{-k} \mod p$ is B-smooth.*

$$h.g^{-k} \equiv \prod_{\ell \leq B} \ell^{e_\ell} \pmod p.$$

- $$\Leftrightarrow log_g(h) \equiv k + \sum_{l \leq B} e_\ell log_g(\ell) \pmod{p-1}.$$

Problem.

How to find $log_g(\ell)$ *for small prime* $l \leq B$ *?*

Algorithm.

For a random i*, comput* $g_i = g^i \pmod{p}$
If g_i *is* B*-smooth, one can factor it as*

$$g_i \equiv \prod_{l \leq B} l^{u_\ell(i)}.$$

$$\Leftrightarrow i = log_g(g_i) \equiv \sum_{l \ leqB} u_\ell(i) log_g(\ell) \pmod{p-1}.$$

If we find more than $\pi(B)$ *equations, then we have a linear system with the unknows* $log_g(\ell)$*, and we can find them.*

Example of the Index Calculus method

Problem.

Let $p = 18443$. Solve the DLP: $37^x \equiv 211 (mod\ 18443)$.

Algorithm.

- *$g = 37$ is a primitive root. Take $B = 5$ then the factor base is $\{2, 3, 5\}$. We will find $x_2 = log_{37} 2, x_3 = log_{37} 2, x_5 = log_{37} 5$ in $\mathbb{F}_{18443}$.*

Example of the Index Calculus method

Problem.

Let $p = 18443$. *Solve the DLP:* $37^x \equiv 211 (mod\ 18443)$.

Algorithm.

- $g = 37$ *is a primitive root. Take* $B = 5$ *then the factor base is* $\{2,3,5\}$. *We will find* $x_2 = log_{37}2, x_3 = log_{37}2, x_5 = log_{37}5$ *in* $\mathbb{F}_{18443}$.
- *Taking random* i *and keep* i *such that* $g^i \mod 18443$ *is a 5-smooth.*
  $g^{12708} \mod 18443 = 2^3.3^4.5$ $\quad$ $g^{11311} \mod 18443 = 2^3.5^2$
  $g^{15400} \mod 18443 = 2^3.3^3.5$ $\quad$ $g^{2731} \mod 18443 = 2^3.3.5^4$

Example of the Index Calculus method

Problem.

Let $p = 18443$. *Solve the DLP:* $37^x \equiv 211 (mod\ 18443)$.

Algorithm.

- $g = 37$ *is a primitive root. Take* $B = 5$ *then the factor base is* $\{2, 3, 5\}$. *We will find* $x_2 = log_{37}2, x_3 = log_{37}2, x_5 = log_{37}5$ *in* $\mathbb{F}_{18443}$.
- *Taking random* i *and keep* i *such that* $g^i \mod 18443$ *is a 5-smooth.*
  $g^{12708} \mod 18443 = 2^3.3^4.5$ $\quad$ $g^{11311} \mod 18443 = 2^3.5^2$
  $g^{15400} \mod 18443 = 2^3.3^3.5$ $\quad$ $g^{2731} \mod 18443 = 2^3.3.5^4$
- *Write the system of linear equations (modulo* $p - 1 = 18442$*).*

$$\begin{array}{rclllll}
12708 & = & 3x_2 & +4x_3 & +x_5 & (mod\ 18442), \\
11311 & = & 3x_2 & & +2x_5 & (mod\ 18442), \\
15400 & = & 3x_2 & +3x_3 & +x_5 & (mod\ 18442), \\
2731 & = & 3x_2 & +x_3 & +4x_5 & (mod\ 18442).
\end{array}$$

- Write $18442 = 2.9221$ Then we solve the above system in $\mathbb{F}_2$ and in $\mathbb{F}_{9221}$. The solution are

$$(x_2, x_3, x_5) \equiv (1, 0, 1) \pmod 2,$$

$$(x_2, x_3, x_5) \equiv (5733, 6529, 6277) \pmod{9221}.$$

- Write $18442 = 2.9221$ Then we solve the above system in $\mathbb{F}_2$ and in $\mathbb{F}_{9221}$. The solution are

$$(x_2, x_3, x_5) \equiv (1, 0, 1) \pmod 2,$$

$$(x_2, x_3, x_5) \equiv (5733, 6529, 6277) \pmod{9221}.$$

- Chinese remainder Theorem:

$$(x_2, x_3, x_5) \equiv (5733, 15750, 6277) \pmod{18442}.$$

- Write $18442 = 2.9221$ Then we solve the above system in $\mathbb{F}_2$ and in $\mathbb{F}_{9221}$. The solution are

$$(x_2, x_3, x_5) \equiv (1, 0, 1) \pmod 2,$$

$$(x_2, x_3, x_5) \equiv (5733, 6529, 6277) \pmod{9221}.$$

- Chinese remainder Theorem:

$$(x_2, x_3, x_5) \equiv (5733, 15750, 6277) \pmod{18442}.$$

- Find k such that $211.37^{-k} \mod 18443$ is a 5-smooth.

$$211.37^{-9549} \equiv 2^5.3^2.5^2 \pmod{18443}.$$

- Write $18442 = 2.9221$ Then we solve the above system in $\mathbb{F}_2$ and in $\mathbb{F}_{9221}$. The solution are

$$(x_2, x_3, x_5) \equiv (1, 0, 1) \pmod 2,$$

$$(x_2, x_3, x_5) \equiv (5733, 6529, 6277) \pmod{9221}.$$

- Chinese remainder Theorem:

$$(x_2, x_3, x_5) \equiv (5733, 15750, 6277) \pmod{18442}.$$

- Find k such that $211.37^{-k} \mod 18443$ is a 5-smooth.

$$211.37^{-9549} \equiv 2^5.3^2.5^2 \pmod{18443}.$$

- $$log_g(211) \equiv 9549 + 5x_2 + 2x_3 + 2x_5 \pmod{18442}.$$

$$\Leftrightarrow log_g(211) \equiv 8500 \pmod{18442}.$$

The solution is 8500.

RSA and Integer Factorization

- The RSA Public Key Cryptosystem
- Pollard's p-1 Factorization
- Factorization via Difference of Squares
- B- smooth number

The RSA Public Key Cryptosystem

Problem.

Alice wants to send a ciphertext to Bob, using Bob's public key.

Algorithm.

Key creation Bob:

- *Choose secret primes p and q.*
- *Choose encryption exponent e with $gcd(e, (p-1)(q-1)) = 1$.*
- *Compute the decryption exponent d: $ed \equiv 1 \pmod{(p-1)(q-1)}$.*
- *Publish the public key: the modulus $N = pq$, the encryption exponent e.*

Encryption Alice:

- *Choose plaintext m.*
- *Compute $c = m^e \mod N$.*
- *Send ciphertext c to Bob.*

Decryption Bob:

- *Compute $m' \equiv c^d \mod N$. This m' is equal to m.*

Example of RSA

Key creation

- Bob chooses: $p = 1223, q = 1987$. He computes $N = pq = 2430101$.
- Bob chooses an encryption exponent $e = 948047$ st. $gcd(e,(p-1)(q-1)) = gcd(948047, 2426892) = 1$.
- Bob solve the equation $ed \equiv 1(\text{mod } (p-1)(q-1)) \Leftrightarrow 848047d \equiv 1\text{mod } 2426892)$ and find $d = 1051235$.

Encryption

- Alice takes $m = 1070777$ satisfying $1 \leq m < N$.
- Alice uses Bob's public key to compute: $c = m^e \mod N = 1070777^{948047} \mod 2430101 = 1473513$.
- Alice send 1473513 to Bob.

Decryption

- Bob computes: $m' = c^d \mod 2430101 = 1473513^{1051235} \mod 2430101 = 1070777$.

Corectness and Complexity

Corectness.

$m' = c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)}.m$

We have $m^{p-1} \equiv 1(\mod p)$ and $m^{q-1} \equiv 1(\mod q)$, then $m^{k(p-1)(q-1)} \equiv 1(\mod pq)$.

$$m' = m^{k(p-1)(q-1)}.m \equiv m (\mathrm{mod}\ N)$$

$$\Leftrightarrow m' \equiv m (\mathrm{mod}\ N) \Leftrightarrow m' = m.$$

Complexity.

- Encryption: easy, in $O(\log e)$ time.
- Decryption: easy, in $O(\log d)$ time.
- Key creation: Compute d by extended euclidean algorithm in $O(\log N)$.

Complexity

Attack:

- If Eve knows p, q, it is OK.
- If Eve knows $(p-1)(q-1)$, it is OK. And this is equivalent to know p and q.
- Eve can find m if she can solve the equation $x^e \equiv c(\text{mod } N)$. What is the complexity of this problem ?
- Factorization problem: Eve try to find p and q knowing $N = pq$.

Pollard's p-1 Factorization algorithm. Idea

Problem.

Knowing that $N = pq$ with large prime numbers p and q. Find p and q.

Main Idea: Find M such that $d = gcd(N, M) \neq 1, N$. Then d will be p.

Algorithm.

1. *Find an integer L st. $(p-1)$ devides L and $(q-1)$ does not.* $\exists i, j, k \neq 0 : L = i(p-1) = j(q-1)j + k$.
2. *Choose randomly a, then*

$$a^L = a^{i(p-1)} = (a^{p-1})^i \equiv 1^i \equiv 1 (mod\ p)$$

$$a^L = a^{j(q-1)+k} = (a^{q-1})^j a^k \equiv 1^i a^k \equiv a^k (mod\ q)$$

If $a^k \neq 1 (mod\ q)$ (hight probability), then $q \nmid (a^L - 1)$.
Then $p = gcd(N, a^L - 1)$.

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?
- If $p-1$ is a product of small primes then $p-1$ divides $n!$ for some value n.

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?
- If $p-1$ is a product of small primes then $p-1$ divides $n!$ for some value n.
- Choose $n = 2, 3, 4, \ldots,$ and compute $gcd(a^{n!} - 1, N)$.

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?
- If $p-1$ is a product of small primes then $p-1$ divides $n!$ for some value n.
- Choose $n = 2, 3, 4, \ldots$, and compute $gcd(a^{n!} - 1, N)$.
- If the gcd is 1, go to the next value of n

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?
- If $p-1$ is a product of small primes then $p-1$ divides $n!$ for some value n.
- Choose $n = 2, 3, 4, \ldots$, and compute $gcd(a^{n!} - 1, N)$.
- If the *gcd* is 1, go to the next value of n
- If the gcd is N, choose another value of a.

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?
- If $p-1$ is a product of small primes then $p-1$ divides $n!$ for some value n.
- Choose $n = 2, 3, 4, \ldots$, and compute $gcd(a^{n!} - 1, N)$.
- If the gcd is 1, go to the next value of n
- If the gcd is N, choose another value of a.
- If the $gcd \neq 1, N$ then it is p.

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?
- If $p-1$ is a product of small primes then $p-1$ divides $n!$ for some value n.
- Choose $n = 2, 3, 4, \ldots$, and compute $gcd(a^{n!} - 1, N)$.
- If the gcd is 1, go to the next value of n
- If the gcd is N, choose another value of a.
- If the $gcd \neq 1, N$ then it is p.
- To compute rapidly $a^{n!}$, we have $a^{n!} = (a^{(n-1)!})^n$. And just consider modulo N.

Pollard's p-1 Factorization algorithm. Idea

- How to find such an integer L: $(p-1)$ devides L and $(q-1)$ does not. ?
- If $p-1$ is a product of small primes then $p-1$ divides $n!$ for some value n.
- Choose $n = 2, 3, 4, \ldots,$ and compute $gcd(a^{n!} - 1, N)$.
- If the gcd is 1, go to the next value of n
- If the gcd is N, choose another value of a.
- If the $gcd \neq 1, N$ then it is p.
- To compute rapidly $a^{n!}$, we have $a^{n!} = (a^{(n-1)!})^n$. And just consider modulo N.
- Compute $a^k \bmod N$ in $O(\log k)$, then $a^{n!} \mod N$ in $O(\log n!) = O(n \log n)$.

Example of Pollard's Factorization algorithm.

1. Input $N = 13927189$.
2. $a = 2$, n begins from 9.

$$2^{9!}-1 \equiv 13867883 \pmod{13927189}, \qquad gcd(2^{9!}-1, 13927189) = 1,$$

$$2^{10!}-1 \equiv 5129508 \pmod{13927189}, \qquad gcd(2^{10!}-1, 13927189) = 1,$$

$$2^{11!}-1 \equiv 4405233 \pmod{13927189}, \qquad gcd(2^{11!}-1, 13927189) = 1,$$

$$2^{12!}-1 \equiv 6680550 \pmod{13927189}, \qquad gcd(2^{12!}-1, 13927189) = 1,$$

$$2^{13!}-1 \equiv 6161077 \pmod{13927189}, \qquad gcd(2^{13!}-1, 13927189) = 1,$$

$$2^{14!}-1 \equiv 879290 \pmod{13927189}, \qquad gcd(2^{14!}-1, 13927189) = 382$$

3. So $p = 3823$. We can check that $p - 1 = 3822 = 2.3.7^2.13$ (this is why 2^{14} works)
4. Then $q = 3643$, and $q - 1 = 2.3.607$, which is not a product of small primes.

Pollard's p-1 Factorization algorithm.

Algorithm.

Input: Integer N to be factorized

1. *Set $a = 2$ (or some other convenient value).*
2. *Loop $j = 2, 3, 4, \ldots$ up to a specified bound*
   1. *Set $a = a^j \mod N$;*
   2. *Compute $d = gcd(a-1, N)$;*
   3. *If $1 < d < N$ then Return d;*
3. *Increment j and loop again at Step 2.*

Conclusion: If $p-1$ or $q-1$ is a product of small primes, then the RSA can be attacked by Pollard's Factorization algorithm.

Factorization via Difference of Squares

Idea

Find a and b such that $N = a^2 - b^2 = (a-b)(a+b)$: a factorization of N.
Looking b from $1, 2, 3, \ldots$ and consider if $N + b^2$ is a perfect square.

Example: $N = 25217$.

$25217 + 1^2 = 25218$,
$25217 + 2^2 = 25221$,
$25217 + 3^2 = 26226$,
$25217 + 4^2 = 25233$,
$25217 + 5^2 = 25242$,
$25217 + 6^2 = 25253$,
$25217 + 7^2 = 26266$,
$25217 + 8^2 = 25281 = 159^2$.
$\Rightarrow 25217 = 159^2 - 8^2 = (159+8)(150-8) = 167.151$

Factorization via Difference of Squares

Idea

Find a and b such that $kN = a^2 - b^2 = (a-b)(a+b)$: factorization of kN.

Example: $N = 203299$. Take b from $1, 2, 3, \ldots$ and test $N + b^2$. Until $b = 100$ it is not OK.

Test $3.N + b^2$

$3.203299 + 1^2 = 609898$,
$3.203299 + 2^2 = 609901$,
$3.203299 + 3^2 = 609906$,
$3.203299 + 4^2 = 609913$,
$3.203299 + 5^2 = 609922$,
$3.203299 + 6^2 = 609933$,
$3.203299 + 7^2 = 609946$,
$3.203299 + 8^2 = 609961 = 781^2$
$\Rightarrow 3.203299 = 178^2 - 8^2 = 789.773$
$gcd(203229, 789) = 263, gcd(203229, 773) = 773$.
Then $N = 263.773$.

A three step factorization procedure

Find a and b such that $a^2 \equiv b^2 (\text{mod } N)$.

Algorithm.

- *Relation Building. Find many integers $a_1, a_2, \ldots, , a_r$ such that $c_i = a_i^2 \mod N$ is a product of small primes.*
- *Elimination. Take a product $c = c_{i_1}, c_{i_2}, \ldots, c_{i_s}$ such that each prime appearing in the product an even power. Then $c = c_{i_1}.c_{i_2}.\ldots.c_{i_s} = b^2$.*
- *GCD Computation. Let $a = a_{i_1}.a_{i_2}.\ldots.a_{i_s}$ Then $a^2 \equiv b^2 (mod\ N)$. Compute $d = gcd(N, a - b)$, d should be a nontrivial factor of N.*

Example: $N = 914387$.
We want to find numbers as products of primes in $\{2, 3, 5, 7\}$

$$1869^2 \equiv 750000 \pmod{914387} \text{ and } 750000 = 2^4.3.5^6$$

$$1909^2 \equiv 901120 \pmod{914387} \text{ and } 901120 = 2^{14}.5.11$$

$$3387^2 \equiv 499125 \pmod{914387} \text{ and } 499125 = 3.5^3.11^3$$

Then
$1869^2.1909^2.3387^2 \equiv 2^{18}.3^2.5^{10}.11^4 = (2^9.3.5^5.11^2)^2 = 580800000^2 \equiv 164255^2 \pmod{914387}$.
Moreover $1869.1909.3387 \equiv 9835 \pmod{914387}$.
Compute $gcd(914387, 9835 - 164255) = 1103$
And $914387 = 1103.829$

A three step factorization procedure

Algorithm.

- *Step 3: GCD Computation.* *Let* $a = a_{i_1}.a_{i_2}.\dots.a_{i_s}$ *Then* $a^2 \equiv b^2 (mod\ N)$. *Compute* $d = gcd(N, a-b)$, *a should be a nontrivial factor of* N.
  It is easy, and the time is $O(\log N)$.
- *Step 2: Elimination.* *Take a product* $c = c_{i_1}.c_{i_2}.\dots.c_{i_s}$ *such that each prime appearing in the product an even power. Then* $c = b^2$.
  Problem: to solve a system of linear equations over the field $\mathbb{F}_2$ *in the specail cse that the corresponding matrix is very sparse.*
- *Step 1: Relation Building.* *Find many integers* $a_1, a_2, \dots, a_r$ *such that* $c_i = a_i^2 \mod N$ *is a product of small primes.*

Step 2: Elimination

Problem.

We have $c_i \equiv a_i^2 \mod N$ and c_i is a product of power of primes in $\{p_1, p_2, \ldots, p_t\}$. We want to find a product of c_i such that each prime appearing in the product an even power.

We have e_{ij} such that

$$\begin{aligned} c_1 &= p_1^{e_{11}} p_2^{e_{12}} \cdots p_t^{e_{1t}}, \\ c_2 &= p_1^{e_{21}} p_2^{e_{22}} \cdots p_t^{e_{2t}}, \\ \ldots \\ c_r &= p_1^{e_{r1}} p_2^{e_{r2}} \cdots p_t^{e_{rt}}. \end{aligned}$$

And we will find $u_1, u_2, \ldots, u_r \in \{0, 1\}$ such that

$$c_1^{u_1}.c_2^{u_2}.\cdots.c_r^{u_r} \text{ ss a perfect square.}$$

$c_1^{u_1}.c_2^{u_2}.\cdots.c_r^{u_r} =$
$p_1^{e_{11}u_1+e_{21}u_2+\cdots+e_{r1}u_r}.p_1^{e_{12}u_1+e_{22}u_2+\cdots+e_{r2}u_r}.\ldots.p_1^{e_{1t}u_1+e_{2t}u_2+\cdots+e_{rt}u_r}.$

Step 2: Elimination

We need

$$\begin{aligned} e_{11}u_1 + e_{21}u_2 + \cdots + e_{r1}u_r &\equiv 0 \pmod 2, \\ e_{12}u_1 + e_{22}u_2 + \cdots + e_{r2}u_r &\equiv 0 \pmod 2, \\ &\cdots \\ e_{1t}u_1 + e_{2t}u_2 + \cdots + e_{rt}u_r &\equiv 0 \pmod 2. \end{aligned}$$

This can be done by standart Gaussian elimination.
Moreover, the matrix is very sparse, then we can solve this system of equations by other more efficient method.

Condition

- The a_i^2 should be greater than N such that $a_i^2 \mod n$ is not trivial.
- The number of variables should be greater tha equal to the number of equations ($r \geq t$) such that there exists solution: the numbers of a_i is greater than the numbers of small primes.

Step 3: Smooth numbers

Definition.

An integer n is called B-smooth if all of its prime factors are less than or equal to B.

Definition.

The function $\psi(X, B)$ counts B-smooth numbers that are smaller than or equal to X.
The function $\pi(B)$ counts prime numbers that are smaller than B.

Condition for Step 2 (Elimination). We find X and B such that $\psi(X, B)$ is greater than $\pi(B)$.
Example $B = 5$,
$5 - smooths : 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30, 32, 36, \ldots$
Not $5 - smooths$:
$7, 11, 13, 14, 17, 19, 21, 23, 26, 28, 29, 31, 33, 34, 35, 37, \ldots$
$\psi(25, 5) = 15$ and $\pi(5) = 3$.

Distribution of smooth numbers

Theorem. (Canfield, Erdos, Pomerance n- 1983)

Fix a number $0 < \epsilon < 1$, and let X and B increase together while satisfying $(lnX)^\epsilon < lnB < (lnX)^{1-\epsilon}$. Let $u = \frac{lnX}{lnB}$. Then

$$\psi(X, B) = X.u^{-u(1+o(1))}.$$

Definition.

$L(X) = e^{\sqrt{(lnX)(lnlnX)}}$. *This fonction is subexponential*

Corollary.

For any fix value of c with $0 < c < 1$,

$$\psi(X, L(X)^c) = X.L(X)^{(-1/2c)(1+o(1))} \text{ as } X \to \infty.$$

Subexponetial running time of the Factorization Algorithm

Proposition.

Let N be a large number, and let $B = L(N)^{1/\sqrt{2}}$.

- *We expect to check approximately $L(N)^{\sqrt{2}}$ random numbers modulo N in order to find at least $\pi(B)$ numbers that are B-smooth.*
- *We expect to check approximetely $L(N)^{\sqrt{2}}$ random numbers of the form $a^2 \mod N$ in order to find enough B-smooth numbers to factor N.*
- *Hence the factorization procedure in three steps should have a subexponential running time.*

A note on subexponential complexity

Definition.
Let $0 \leq a \leq 1$ *and* $c \in \mathbb{R}^+$. *The subexponential function for the parameters* a *and* c *is*

$$L_N(a, c) = exp(c \log(N)^a) \log(\log(N))^{1-a}).$$

A complexity $O(L_N(a, c))$ *with* $0 < a < 1$ *is called subexponential.*

- Note:
  If $a = 0$ then $L_N(0, c) = \log(N)^c$: polynomial
  If $a = 1$ then $L_N(1, c) = N^c$: exponential.

Some References

1. An Introduction to Mathematical Cryptography. Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman. Springer-Verlag. 2008

2. Mathematics of Public Key Cryptography. Steven Galbraith, available from http://www.isg.rhul.ac.uk/sdg/crypto-book/