



Chyba v digitálnom podpise v slovenských OP

Marek Sys, Matus Nemec, Petr Svenda, Dusan Klinec, Vashek Matyas

Masaryk University, BRNO

14. decembra 2017



Primitive Root - Primitívny koreň

Let G be a multiplicative group and let $g \in G$ be an element of G . Let N be the smallest exponent such that $g^N = 1$. We will say that N is **the multiplicative order of g** , resp. g is an element of multiplicative order N .

A **primitive root** of a prime p is an integer $g \neq 1$ such that $g \pmod{p}$ has multiplicative order $p - 1$, i.e. $g^{p-1} \equiv 1 \pmod{p}$.

More generally, if $GCD(g, n) = 1$ (g and n are relatively prime) and g is of multiplicative order $\phi(n)$ modulo n (– i.e. $g^{\phi(n)} \equiv 1 \pmod{n}$ where $\phi(n)$ is the Euler's totient function), then g is called a **primitive root of n** .

The first definition is a special case of the second since $\phi(p) = p - 1$ for p a prime.

Example.

Number 4 is not a primitive root of 5 resp. $\mathbf{Z}_5 - \{0\}$ since multiplicative order of 4 is 2 because $4^2 = 16 \equiv 1 \pmod{5}$.

Number 3 is a primitive root of \mathbf{Z}_5 since

$$3^{\phi(5)} = 3^{5-1} = 3^4 = 81 \equiv 1 \pmod{5}$$

while $3^2 = 9 \equiv 4 \pmod{5}$, $3^3 = 27 \equiv 2 \pmod{5}$.

Therefore $\phi(5) = 4$ is the smallest exponent such that $3^4 \equiv 1 \pmod{5}$.

Number 2 is a primitive root of \mathbf{Z}_5 since

$$2^{\phi(5)} = 2^{5-1} = 2^4 = 16 \equiv 1 \pmod{5}$$

while $2^2 = 4 \equiv 4 \pmod{5}$, $2^3 = 8 \equiv 3 \pmod{5}$.

Therefore $\phi(5) = 4$ is the smallest exponent such that $2^4 \equiv 1 \pmod{5}$.

Not every n has a primitive root !!!

Example.

Number $n = 8$ has no primitive root.

$$\phi(8) = 4$$

For no even number $g \in \mathbf{Z}_8$ and no $a > 1$ it holds $g^a \equiv 1 \pmod{8}$.

Multiplicative orders of 3, 5 and 7 are all equal to 2 since

$$3^2 = 9 \equiv 1 \pmod{8}, \quad 5^2 = 25 \equiv 1 \pmod{8}, \quad 7^2 = 49 \equiv 1 \pmod{8}.$$

\mathbf{Z}_8 does not contain an element g with multiplicative order equal to $\phi(8) = 4$.



Primitive Root - Primitívny koreň

If n has a primitive root, then it has exactly $\phi(\phi(n))$ of them, which means that if p is a prime number, then there are exactly $\phi(p - 1)$ incongruent primitive roots of p .

For $n=1, 2, \dots$, the first few values of $\phi(\phi(n))$ are

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	1	1	2	1	2	2	2	2	4	2	4	2	4	4	8

A number n has a primitive root

if it is of the form $2, 4, p^a$, or $2p^a$, where p is an odd prime and $a \geq 1$.

The first few n for which primitive roots exist are

2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 17, 18, 19, 22, ...

so the number of primitive root of order n for $n=1, 2, \dots$ are

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	1	1	2	1	2	0	2	2	4	0	4	2	0	0



Primitive Root - Primitívny koreň

Here is table of the primitive roots for the first few n for which a primitive root exists:

n	$g(n)$
2	1
$3 = p^1$	2
4	3
$5 = 5^1$	2, 3
$6 = 2 * 3^1$	5
$7 = 7^1$	3, 5
$9 = 3^2$	2, 5
$10 = 2 * 5^1$	3, 7
$11 = 11^1$	2, 6, 7, 8
$13 = 13^1$	2, 6, 7, 11

Let G be a multiplicative group and let $g \in G$, $h \in G$ are elements of G .

Discrete Logarithm Problem – DLP is to find a natural number x such that

$$g^x = h \tag{1}$$



Baby step – Giant step algorithm

Let $G = (\mathbf{Z}_p - \{0\}, \otimes)$ be a group and let $g \in G$ be an element of order $N \geq 2$. Let $h \in G$.

Input:

h, g, p , solve equation $h \equiv g^x \pmod{p}$.

Step 1.

Let $n = \lceil \sqrt{N-1} \rceil$ (therefore $n > \sqrt{N-1}$).

Step 2.

Create 2 lists.

$$g^0, g^1, g^2, \dots, g^i, \dots, g^{n-1}$$
$$hg^{-0n}, hg^{-1n}, hg^{-2n}, hg^{-3n}, \dots, hg^{-jn}, \dots, hg^{-(n-1)n}$$

Step 3.

Find a match in the 2 lists. Say $g^i = hg^{-jn}$.

Then $x = i + jn$ is the solution.

This algorithm solves the DLP in $O(\sqrt{N})$ steps.

Baby step – Giant step algorithm

Let us solve the following digital logarithm problem

$$3^x \equiv 57 \pmod{113} \quad (2)$$

Then $g = 3$, $p = 113$, $h = 57$, $\sqrt{113} = 10.63$ and $n = \lceil 10.63 \rceil = 11$.

i	0	1	2	3	4	5	6	7	8	9	10
$3^i \pmod p$	1	3	9	27	81	17	51	40	7	21	63
$57 \cdot 3^{-i \cdot n} \pmod p$	57	29	100	37	112	55	26	39	2	3	

Since $3^{-1} \equiv 38 \pmod p$, the last row of previous table can be calculated by formula $57 * 38^{i*n} \pmod p$.

$$\begin{aligned} 3^1 &= 57 * 3^{(-9*11)} \\ (3^1) * (3^{(9*11)}) &= 57 \\ 3^{1+(9*11)} &= 57 \\ 3^{100} &= 57 \\ x &= 100 \end{aligned}$$

Let n_1, n_2, \dots, n_k be integers greater than 1, which are often called moduli or divisors.

Let us denote by N the product of the n_i , i.e.,:

$$N = n_1 \cdot n_2 \cdot \dots \cdot n_k.$$

The Chinese remainder theorem. Let n_i are pairwise coprime, let a_1, a_2, \dots, a_k are integers such that $0 \leq a_i < n_i$ for every i .

Then there is one and only one integer x , such that $0 \leq x < N$ and such that the remainder of the integer division of x by n_i is a_i for every i .



Chinese remainder theorem

This may be restated as follows in term of congruences:

Let n_1, n_2, \dots, n_k be integers greater than 1.

Let us denote by N the product of the n_i , i.e.,

$$N = n_1 \cdot n_2 \cdot \dots \cdot n_k.$$

If the n_i are pairwise coprime, and if a_1, a_2, \dots, a_k are any integers, then there exists an integer x , $0 \leq x < N$ such that

$$x \equiv a_1 \pmod{n_1}$$

$$x \equiv a_2 \pmod{n_2}$$

$$\vdots$$

$$x \equiv a_k \pmod{n_k}$$

There exists a polynomial algorithm for computing such x .



Chinese remainder theorem

We want to solve first a system with two equations:

$$x \equiv a_1 \pmod{n_1} \quad (3)$$

$$x \equiv a_2 \pmod{n_2}, \quad (4)$$

First solve the following equation with unknown r using extended Euklid's Algorithm.

$$n_1 * r \equiv 1 \pmod{n_2}.$$

This equation has a solution since n_1 and n_2 are coprime.

$$\text{Then } n_1 \cdot r = n_2 \cdot s + 1 \text{ for some integer } s$$

$$n_1 \cdot r - n_2 \cdot s = 1$$

We have after substituting m_1 for r and m_2 for $-s$

$$n_1 m_1 + n_2 m_2 = 1. \quad (5)$$

Then the solution of the system (3), (4) is

$$x = a_1 n_2 m_2 + a_2 n_1 m_1 \quad (6)$$

Indeed:

$$\begin{aligned} x &= a_1 n_2 m_2 + a_2 n_1 m_1 = a_1(1 - n_1 m_1) + a_2 n_1 m_1 = \\ &= a_1 - a_1 n_1 m_1 + a_2 n_1 m_1 = a_1 + (a_2 - a_1) n_1 m_1 \end{aligned} \quad (7)$$

what implies that $x \equiv a_1 \pmod{n_1}$.



Chinese remainder theorem

Let n_1, n_2, \dots, n_k be integers greater than 1.

Let us denote by N the product of the n_i , i.e.:

$$N = n_1 \cdot n_2 \cdot \dots \cdot n_k.$$

If the n_i are pairwise coprime, and if a_1, a_2, \dots, a_k are any integers, then there exists an integer x , $0 \leq x < N$ such that

$$\begin{aligned}x &\equiv a_1 \pmod{n_1} \\x &\equiv a_2 \pmod{n_2} \\&\vdots \\x &\equiv a_k \pmod{n_k}\end{aligned}$$

Define $b_i = \frac{N}{n_i}$, set $c_i = b_i^{-1} \pmod{n_i}$.

$$x \equiv \sum_{i=1}^k a_i b_i c_i \pmod{N} \quad (8)$$

is the solution of given system of equations.

Pohling – Hellmann Algorithm

Let G be a group, let $g \in G$ an element of order N and suppose that N factors into product of prime powers as

$$N = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_t^{e_t}. \quad (9)$$

Pohlig – Hellman Algorithm

Input:

h, g, p , solve equation $h \equiv g^x \pmod{p}$.

Step 1.

For each $i, 1 \leq i \leq t$, set $g_i \equiv g^{N/p^{e_i}}$ and $h_i \equiv h^{N/p^{e_i}}$.

For every i use Baby step – Giant step algorithm to solve equation

$$g_i^{y_i} = h_i. \quad (10)$$

Step 2.

For every $i, 1 \leq i \leq t$ use the Chinese Remainder Theorem to solve system of equations

$$\begin{aligned} x &\equiv y_1 \pmod{p_1^{e_1}} \\ x &\equiv y_2 \pmod{p_2^{e_2}} \\ &\vdots \\ x &\equiv y_t \pmod{p_t^{e_t}} \end{aligned}$$

Return x .



RSA Algorithm – Key Generation Procedure

Alice executes the following procedure to calculate her pair of RSA keys

- 1 Select two distinct large primes p and q . Both primes have to be kept in secrecy.
- 2 Compute $N = p \cdot q$ and $\phi(N) = (p - 1) \cdot (q - 1)$. $\phi(N)$ should be secret.
- 3 Choose a public exponent $e < \phi(N)$, e coprime to $\phi(N)$.
- 4 Compute the private exponent d as $e \cdot d \equiv 1 \pmod{\phi(N)}$.
- 5 Pair $(e; N)$ is the public key – it can be published.
- 6 Pair $(d; N)$ serves as the secret private key.
- 7 Bob (or arbitrary participant in secret communication) enciphers his message $x < N$ as $y = x^e \pmod{N}$.
- 8 Alice deciphers ciphertext y as $x = y^d \pmod{N}$.

Primes p , q should be large – from 512 to 2048 bits.

How to find such numbers – choose a large number at random and test it for primality.



RSA Algorithm – Key Generation Procedure

Probability primality test

Fermat test

- If $c^{(M-1)} \not\equiv 1 \pmod{M}$ for some c , then M is definitely composed number.
- If $c^{(M-1)} \equiv 1 \pmod{M}$ for sufficiently many numbers c , then M is prime with great probability.

Phill Zimmermann used in PGP the following procedure for finding whether M is a prime:

- Discarded M if it failed to get through test based on dividing by all 16-bit primes
- Applied Fermat's test for four values of the number c .

Drawback of Fermat test: There are liars called Carmichael numbers.

Carmichael's number – is a composite number M , such that for all $c < M$, c coprime to M it holds $c^{M-1} \equiv 1 \pmod{M}$.

Miller-Rabin test can say that the examined number is surely composite or prime with probability $(1 - \frac{1}{4^t})$.



RSALib – Key Generation Procedure

All RSA primes (as well as the moduli) generated by the RSALib have the following form:

$$p = k * M + (65537^a \pmod M) \quad (11)$$

where integers k and a are unknown.

The integer M is known and equal to some primorial

$M = P_n\#$ – the product of the first n successive primes

$$P_n\# = \prod_{i=1}^n P_i = 2 * 3 * 5 * 7 * 11 * \dots * P_n.$$

The value of M is related to the key size.

The value $n = 39$ (i.e., $M = 2 * 3 * \dots * 167$) is used to generate primes for an RSA key with a key size within the [512 – 960] interval.

n	39	71	126	225
key size	[512 – 960]	[992 – 1952]	[1984 – 3936]	[3968 – 4096]
$P_n\#$	$P_{39}\# = 167\#$	$P_{71}\# = 353\#$	$P_{126}\# = 701\#$	$P_{225}\# = 1427\#$
size of M	219 b	475 b	971 b	1962 b



RSAlib – Key Generation Procedure

RSA primes differ only in their values of a and k for keys of the same size.

The most important property of the keys is that the size of M is large and almost comparable to the size of the prime p (e.g., M has 219 bits for the 256-bit prime p used for 512-bit RSA keys).

Since M is large, the sizes of k and a are small (e.g., k has $256 - 219 = 37$ bits and a has 62 bits for 512-bit RSA).

Hence, the resulting RSA primes suffer from a significant loss of entropy (e.g., a prime used in 512-bit RSA has only $99 = 37 + 62$ bits of entropy).

The pool from which primes are randomly generated is reduced (e.g., from 2^{256} to 2^{99} for 512-bit RSA).

Let $N = p \cdot q$.

$$N = \underbrace{(k * M + 65537^a \bmod M)}_p * \underbrace{(l * M + 65537^b \bmod M)}_q \quad (12)$$

$$N \equiv 65537^{(a+b)} \equiv 65537^c \bmod M \quad (13)$$

If there exists the discrete logarithm

$$c = \log_{65537} N \bmod M \quad (14)$$

then modulus N was probably generated by RSALib.

Discrete logarithm is a hard problem, however we can use Pohling-Hellman algorithm effectively in special cases where M is a smooth number.

Size of multiplicative group \mathbf{Z}_M^* is $\phi(M)$. Since M is a smooth number – having only small factors, also $\phi(M)$ is a smooth number.

The order of subgroup $G = [65537]$ of \mathbf{Z}_M^* (subgroup generated by 65537) is a divisor of $\phi(M)$ and therefore smooth number, too. Therefore Pohling-Hellmann algorithm gives us result within miliseconds.

The probability that that a random 512-bit modulus will be incorrectly detected as the RSALib modulus is 2^{-154} . This probability is even smaller for larger keys.

Principle of this attack is to apply Coppersmith's algorithm, which was originally proposed to find small roots of univariate modular equations.

Coppersmith showed how to use his algorithm to factorize RSA modulus, when high bits of p or q are known.

Let

$$N = \underbrace{(k * M + 65537^a \pmod{M})}_p * \underbrace{(l * M + 65537^b \pmod{M})}_q.$$

A naive approach would iterate over different options of $65537^a \pmod{M}$, treating the value of $65537^a \pmod{M}$ as known bits and apply Coppersmith's algorithm to find unknown k .


Complexity of this approach showed to be too large.

In order to optimize the naive method, the authors replaced M by another M' such that:

- primes (p, q) are still of the form $p = k *' M + 65537^{a'} \pmod{M'}$,
 $q = l' * M + 65537^{b'} \pmod{M'}$ – i.e. M' must be a divisor of M
- Coppersmith's algorithm will find k for correct guess of a – enough bits must be known ($\log_2(M') > \log_2(N)/4$)
- overall time of the factorization will be minimal – number of attempts ($ord_{M'}(65537)$) and time per attempt (running time of Coppersmith's algorithm) should result in a minimal time.

There is a trade-off between the number of attempts and the computational time per attempt as Coppersmith's algorithm runs faster when more bits are known.

Running time of Coppersmith's algorithm can be estimated by M' and by two parameters m and t .



Coppersmith's algorithm

Coppersmith's algorithm will solve the equation

$$f(x) = x * M' + (65537^{a'} \bmod M') = 0 \bmod N \quad (15)$$

resp. its equivalent form

$$x + (M'^{-1} \bmod N) * (65537^{a'} \bmod M') = 0 \bmod N \quad (16)$$

Coppersmith's algorithm is based on the LLL method of lattice base reduction.¹

¹There are two meanings of the term lattice. By the first one a lattice is a partially ordered set L in which each two-element subset $\{a, b\} \in L$ has a join (i.e. least upper bound) and a meet (i.e. greatest lower bound), denoted by $a \vee b$ and $a \wedge b$. By the second meaning a lattice is subgroup Λ of \mathbf{R}^n

$$\Lambda = \left\{ \sum_{i=1}^n a_i \mathbf{v}_i \mid a_i \in \mathbf{Z} \right\}$$

where $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$ is a basis of linear space \mathbf{R}^n .

Coppersmith's algorithm makes use of the second definition of a lattice and lattice base reduction problem – finding an equivalent basis with short vectors. A. Lenstra, H. Lenstra and L. Lovász proposed LLL lattice base reduction algorithm solving lattice base reduction problem in polynomial time.

Input: N, M', m, t

Output: p – factor of N

$$c' = \log_{65537} N \bmod M'$$

$$\text{ord}' = \text{ord}_{M'}(65537)$$

forall $a' \in \left[\frac{c'}{2}, \frac{c' + \text{ord}'}{2} \right]$ **do**

$$f(x) = (x + (M'^{-1} \bmod N) * (65537^{a'} \bmod M')) \bmod N$$

$$(\beta, X) = (.5, 2 * N^{\beta/M'})$$

$$k' = \text{Coppersmith}(f(x), N, \beta, m, t, X)$$

$$p = k' * M' + (65537^{a'}) \bmod M'$$

if $N \bmod p = 0$ **then return** p

end



Optimized method

Key size	M	Size of M	Size of M'	Naive BF #	Our BF#	Time per attempt	Worst case
				attempts $ord_M(65537)$	attempts $ord_{M'}(65537)$		
				2	2		
512 b	$P_{39}\# = 167\#$	219.19 b	140.77 b	$2^{61.09}$	$2^{19.20}$	11.6 ms	1.93 CPU hours
1024 b	$P_{71}\# = 353\#$	474.92 b	285.19 b	$2^{133.73}$	$2^{29.04}$	15.2 ms	97.1 CPU days
2048 b	$P_{126}\# = 701\#$	970.96 b	552.50 b	$2^{254.78}$	$2^{34.29}$	212 ms	140.8 CPU years
3072 b	$P_{126}\# = 701\#$	970.96 b	783.62 b	$2^{254.78}$	$2^{99.29}$	1159 sec	$2.84 * 10^{25}$ years
4096 b	$P_{225}\# = 1427\#$	1962.19 b	1098.42 b	$2^{433.69}$	$2^{55.05}$	1086 ms	$1.28 * 10^9$ years

Table 1: Overview of the used parameters (original M and optimized M')