Workshop on Cryptology
September 24–25, 2005
IEEE Student Branch, Kharagpur

Dr. Abhijit Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Kharagpur 721302, India

Part I

Introduction to number theory

Common sets

$$\begin{aligned}
\mathbb{N} &= \{1, 2, 3, \ldots\} \quad \text{(Natural numbers)} \\
\mathbb{N}_0 &= \{0, 1, 2, 3, \ldots\} \quad \text{(Non-negative integers)} \\
\mathbb{Z} &= \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\} \quad \text{(Integers)} \\
\mathbb{P} &= \{2, 3, 5, 7, 11, 13, \ldots\} \quad \text{(Primes)}
\end{aligned}$$

Divisibility: $a \mid b$ if $b = ac$ for some $c \in \mathbb{Z}$.

Corollary: If $a \mid b$, then $|a| \leqslant |b|$.

Theorem: There are infinitely many primes.

Euclidean division: Let $a, b \in \mathbb{Z}$ with $b > 0$. There exist unique $q, r \in \mathbb{Z}$ with $a = qb + r$ and $0 \leqslant r < b$.

Notations: $q = a \text{ quot } b$, $r = a \text{ rem } b$.

GCD (Greatest common divisor)

Let $a, b \in \mathbb{Z}$, not both zero. Then $d \in \mathbb{N}$ is called the gcd of a and b, if:

(1) $d \mid a$ and $d \mid b$.

(2) If $d' \mid a$ and $d' \mid b$, then $d' \mid d$.

We denote $d = \gcd(a, b)$.

Euclidean gcd: $\gcd(a, b) = \gcd(b, a \text{ rem } b)$ (for $b > 0$).

Extended gcd: Let $a, b \in \mathbb{Z}$, not both zero. There exist $u, v \in \mathbb{Z}$ such that

$$\gcd(a, b) = ua + vb.$$

Example

$$\begin{aligned}
899 &= 2 \times 319 + 261, \\
319 &= 1 \times 261 + 58, \\
261 &= 4 \times 58 + 29, \\
58 &= 2 \times 29.
\end{aligned}$$

$\gcd(899, 319) = 29$

Extended gcd computation

$$\begin{aligned}
29 &= 261 - 4 \times 58 \\
&= 261 - 4 \times (319 - 1 \times 261) = (-4) \times 319 + 5 \times 261 \\
&= (-4) \times 319 + 5 \times (899 - 2 \times 319) = 5 \times 899 + (-14) \times 319.
\end{aligned}$$

Modular arithmetic

Let $n \in \mathbb{N}$. Define $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$.

Addition: $a + b \pmod{n} = \begin{cases} a+b & \text{if } a+b < n \\ a+b-n & \text{if } a+b \geqslant n \end{cases}$

Subtraction: $a - b \pmod{n} = \begin{cases} a-b & \text{if } a-b \geqslant 0 \\ a-b+n & \text{if } a-b < 0 \end{cases}$

Multiplication: $ab \pmod{n} = ab \operatorname{rem} n$.

Inverse: $a \in \mathbb{Z}_n$ is called *invertible* modulo n if $ua \operatorname{rem} n = 1$ for some $u \in \mathbb{Z}_n$.

Theorem: $a \in \mathbb{Z}_n$ is invertible modulo n if and only if $\gcd(a, n) = 1$. In this case extended gcd gives $ua + vn = 1$. We may take $0 \leqslant u < n$. We have $u = a^{-1} \pmod{n}$.

Example of modular arithmetic

Take $n = 257$, $a = 127$, $b = 217$.

Addition: $a + b = 344 > 257$, so $a + b \pmod{n} = 344 - 257 = 87$.

Subtraction: $a - b = -90 < 0$, so $a - b \pmod{n} = -90 + 257 = 167$.

Multiplication: $ab \pmod{n} = (127 \times 217) \text{ rem } 257 = 27559 \text{ rem } 257 = 60$.

Inverse: $\gcd(b, n) = 1 = (-45)b + 38n$, so $b^{-1} \pmod{n} = -45 + 257 = 212$.

Division: $a/b \pmod{n} = ab^{-1} \pmod{n} = (127 \times 212) \text{ rem } 257 = 196$.

Modular exponentiation

Let $n \in \mathbb{N}$, $a \in \mathbb{Z}_n$ and $e \in \mathbb{N}_0$. To compute $a^e \pmod{n}$.

Slow algorithm

Compute $a, a^2, a^3, \ldots, a^e$ successively by multiplying with a modulo n.

Example: $n = 257$, $a = 127$, $e = 217$.

$$
\begin{aligned}
a^2 &= a \times a = 195 \pmod{n}, \\
a^3 &= a^2 \times a = 195 \times 127 = 93 \pmod{n}, \\
a^4 &= a^3 \times a = 93 \times 127 = 246 \pmod{n}, \\
&\cdots \\
a^{216} &= a^{215} \times a = 131 \times 127 = 189 \pmod{n}, \\
a^{217} &= a^{216} \times a = 189 \times 127 = 102 \pmod{n}.
\end{aligned}
$$

Modular exponentiation (contd)

Fast algorithm

Binary representation: $e = (e_{l-1}e_{l-2}\ldots e_1e_0)_2 = e_{l-1}2^{l-1} + e_{l-2}2^{l-2} + \cdots + e_1 2^1 + e_0 2^0$.

$$a^e = \left(a^{2^{l-1}}\right)^{e_{l-1}} \left(a^{2^{l-2}}\right)^{e_{l-2}} \cdots \left(a^{2^1}\right)^{e_1} \left(a^{2^0}\right)^{e_0} \pmod{n}.$$

Compute $a, a^2, a^{2^2}, a^{2^3}, \ldots, a^{2^{l-1}}$ and multiply those a^{2^i} modulo n for which $e_i = 1$. Also for $i \geqslant 1$, we have $a^{2^i} = \left(a^{2^{i-1}}\right)^2 \pmod{n}$.

Example: $n = 257$, $a = 127$, $e = 217$.

$e = (11011001)_2 = 2^7 + 2^6 + 2^4 + 2^3 + 2^0$. So $a^e = a^{2^7} a^{2^6} a^{2^4} a^{2^3} a^{2^0} \pmod{n}$.

$a^2 = 195 \pmod{n}$, $a^{2^2} = (195)^2 = 246 \pmod{n}$, $a^{2^3} = (246)^2 = 121 \pmod{n}$, $a^{2^4} = (121)^2 = 249 \pmod{n}$, $a^{2^5} = (249)^2 = 64 \pmod{n}$, $a^{2^6} = (64)^2 = 241 \pmod{n}$ and $a^{2^7} = (241)^2 = 256 \pmod{n}$.

$a^e = 256 \times 241 \times 249 \times 121 \times 127 = 102 \pmod{n}$.

Euler totient function

Let $n \in \mathbb{N}$. Define $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$. Thus $\mathbb{Z}_n^*$ is the set of all elements of $\mathbb{Z}_n$ that are invertible modulo n.

Call $\phi(n) = |\mathbb{Z}_n^*|$.

Example: If p is a prime, then $\phi(p) = p - 1$.

Example: $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$. We have $\gcd(0, 6) = 6$, $\gcd(1, 6) = 1$, $\gcd(2, 6) = 2$, $\gcd(3, 6) = 3$, $\gcd(4, 6) = 2$, and $\gcd(5, 6) = 1$. So $\mathbb{Z}_6^* = \{1, 5\}$, i.e., $\phi(6) = 2$.

Theorem: Let $n = p_1^{e_1} \cdots p_r^{e_r}$ with distinct primes $p_i \in \mathbb{P}$ and with $e_i \in \mathbb{N}$. Then

$$\phi(n) = n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_r}\right) = n \prod_{p \,|\, n} \left(1 - \frac{1}{p}\right).$$

Fermat's little theorem: Let $p \in \mathbb{P}$ and $a \in \mathbb{Z}$ with $p \not| \, a$. Then $a^{p-1} = 1 \pmod{p}$.

Euler's theorem: Let $n \in \mathbb{N}$ and $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$. Then $a^{\phi(n)} = 1 \pmod{n}$.

Multiplicative order

Let $n \in \mathbb{N}$ and $a \in \mathbb{Z}_n^*$. Define $\operatorname{ord}_n a$ to be the smallest of the *positive* integers h for which $a^h = 1 \pmod n$.

Example: $n = 17$, $a = 2$. $a^1 = 2 \pmod n$, $a^2 = 4 \pmod n$, $a^3 = 8 \pmod n$, $a^4 = 16 \pmod n$, $a^5 = 15 \pmod n$, $a^6 = 13 \pmod n$, $a^7 = 9 \pmod n$, and $a^8 = 1 \pmod n$. So $\operatorname{ord}_{17} 2 = 8$.

Theorem: $a^k = 1 \pmod n$ if and only if $\operatorname{ord}_n a \mid k$.

Theorem: Let $h = \operatorname{ord}_n a$. Then $\operatorname{ord}_n a^k = h/\gcd(h,k)$.

Theorem: $\operatorname{ord}_n a \mid \phi(n)$.

Primitive root

If $\operatorname{ord}_n a = \phi(n)$, then a is called a primitive root modulo n.

Theorem (Gauss): An integer $n > 1$ has a primitive root if and only if $n = 2, 4, p^e, 2p^e$, where p is an odd prime and $e \in \mathbb{N}$.

Example: 3 is a primitive root modulo the prime $n = 17$:

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$3^k \pmod{17}$	1	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

Example: $n = 2 \times 3^2 = 18$ has a primitive root 5 with order $\phi(18) = 6$:

k	0	1	2	3	4	5	6
$5^k \pmod{18}$	1	5	7	17	13	11	1

Example: $n = 20 = 2^2 \times 5$ does not have a primitive root. We have $\phi(20) = 8$, and the orders of the elements of $\mathbb{Z}_{20}^*$ are $\operatorname{ord}_{20} 1 = 1$, $\operatorname{ord}_{20} 3 = \operatorname{ord}_{20} 7 = \operatorname{ord}_{20} 13 = \operatorname{ord}_{20} 17 = 4$, and $\operatorname{ord}_{20} 9 = \operatorname{ord}_{20} 19 = 2$.

Discrete logarithm

Let $p \in \mathbb{P}$, g a primitive root modulo p, and $a \in \{1, 2, \ldots, p-1\}$. Then there exists a unique integer $x \in \{0, 1, 2, \ldots, p-2\}$ such that $g^x = a \pmod{p}$. We call x the *index* or *discrete logarithm* of a to the base g. We denote this by $x = \operatorname{ind}_g a$.

Indices follow arithmetic modulo $p-1$.

$$\begin{aligned} \operatorname{ind}_g(ab) &= \operatorname{ind}_g a + \operatorname{ind}_g b \pmod{p-1}, \\ \operatorname{ind}_g(a^e) &= e \operatorname{ind}_g a \pmod{p-1}. \end{aligned}$$

Example: Take $p = 17$ and $g = 3$.

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\operatorname{ind}_3 a$	0	14	1	12	5	15	11	10	2	3	7	13	4	9	6	8

$\operatorname{ind}_3 6 = 15$ and $\operatorname{ind}_3 11 = 7$. Since $6 \times 11 = 15 \pmod{17}$, we have $\operatorname{ind}_3 15 = \operatorname{ind}_3 6 + \operatorname{ind}_3 11 = 15 + 7 = 6 \pmod{16}$.

Part II

Public key cryptography

Intractable problems

Integer factorization problem (IFP): Given $n \in \mathbb{N}$ compute the complete prime factorization of n. Suppose there is an algorithm A that computes a non-trivial factor of n. We can use A repeatedly in order to compute the complete factorization of n. If $n = pq$ (with $p, q \in \mathbb{P}$), then computing a single prime factor (p or q) of n suffices.

Example

Input: $n = 85067$.
Output: $85067 = 257 \times 331$.

Discrete logarithm problem (DLP): Let $p \in \mathbb{P}$ and g a primitive root modulo p. Given $a \in \mathbb{Z}_p^*$, compute $\operatorname{ind}_g a$.

Example

Input: $p = 17$, $g = 3$, $a = 11$.
Output: $\operatorname{ind}_g a = 7$.

Intractable problems (contd)

IFP and DLP are believed to be computationally very difficult.

The best known algorithms for IFP and DLP are subexponential.

IFP is the inverse of the integer multiplication problem.

DLP is the inverse of the modular exponentiation problem.

Integer multiplication and modular exponentiation are easy computational problems. They are believed to be one-way functions.

There is, however, no proof that IFP and DLP must be difficult.

Efficient quantum algorithms exist for solving IFP and DLP.

IFP and DLP are believed to be computationally equivalent.

Intractable problems (contd)

Diffie-Hellman problem (DHP): Let $p \in \mathbb{P}$ and g a primitive root modulo p. Given g^x and g^y modulo p, compute g^{xy} modulo p.

Example

Input: $p = 17$, $g = 3$, $g^x = 11 \pmod{p}$ and $g^y = 13 \pmod{p}$.
Output: $g^{xy} = 4 \pmod{p}$.
($x = 7$, $y = 4$, i.e., $xy = 28 = 12 \pmod{p-1}$, i.e., $g^{xy} = 3^{12} = 4 \pmod{p}$.)

DHP is another believably difficult computational problem.

If DLP can be solved, then DHP can be solved ($g^{xy} = (g^x)^y$).

The converse is believed to be true.

RSA encryption

Key generation

The recipient generates two random large primes p, q, computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, finds a random integer e with $\gcd(e, \phi(n)) = 1$, and determines an integer d with $ed = 1 \pmod{\phi(n)}$.

Public key: (n, e).
Private key: (n, d).

Encryption

Input: Plaintext message $m \in \mathbb{Z}_n$ and the recipient's public key (n, e).
Output: Ciphertext message $c = m^e \pmod{n}$.

Decryption

Input: Ciphertext message c and the recipient's private key (n, d).
Output: Plaintext message $m = c^d \pmod{n}$.

Example of RSA encryption

Let $p = 257$, $q = 331$, so that $n = pq = 85067$ and $\phi(n) = (p-1)(q-1) = 84480$. Take $e = 7$, so that $d = e^{-1} = 60343 \pmod{\phi(n)}$.

<u>Public key</u>: $(85067, 7)$.
<u>Private key</u>: $(85067, 60343)$.

Let $m = 34152$. Then $c = m^e = (34152)^7 = 53384 \pmod{n}$.

Recover $m = c^d = (53384)^{60343} = 34512 \pmod{n}$.

Decryption by an exponent d' other than d does not give back m. For example, take $d' = 38367$. We have $m' = c^{d'} = (53384)^{38367} = 71303 \pmod{n}$.

Why RSA works?

Assume that $m \in \mathbb{Z}_n^*$. By Euler's theorem $m^{\phi(n)} = 1 \pmod{n}$.

Now $ed = 1 \pmod{\phi(n)}$, i.e., $ed = 1 + k\phi(n)$ for some integer k. Therefore,

$$c^d = m^{ed} = m^{1+k\phi(n)} = m \times \left(m^{\phi(n)}\right)^k = m \times (1)^k = m \pmod{n}.$$

Note: The message can be recovered uniquely even when $m \notin \mathbb{Z}_n^*$.

RSA signature

Key generation

The signer generates two random large primes p, q, computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, finds a random integer e with $\gcd(e, \phi(n)) = 1$, and determines an integer d with $ed = 1 \pmod{\phi(n)}$.

<u>Public key</u>: (n, e).
<u>Private key</u>: (n, d).

Signing

<u>Input</u>: Message $m \in \mathbb{Z}_n$ to be signed and the signer's private key (n, d).
<u>Output</u>: The signature (m, s), where $s = m^d \pmod{n}$.

Verification

<u>Input</u>: Signed message (m, s) and the signer's public key (n, e).
<u>Output</u>: "Signature verified" if $s^e = m \pmod{n}$,
"Signature not verified" if $s^e \neq m \pmod{n}$.

Example of RSA signature

Let $p = 257$, $q = 331$, so that $m = pq = 85067$ and $\phi(n) = (p-1)(q-1) = 84480$. Take $e = 19823$, so that $d = e^{-1} = 71567 \pmod{\phi(n)}$.

Public key: $(85067, 19823)$.
Private key: $(85067, 71567)$.

Let $m = 3759$ be the message to be signed. Generate $s = m^d = 13728 \pmod{n}$. The signed message is $(3759, 13728)$.

Verification of $(m, s) = (3759, 13728)$ involves the computation of $s^e = (13728)^{19823} = 3759 \pmod{n}$. Since this equals m, the signature is verified.

Verification of a forged signature $(m, s) = (3759, 42954)$ gives $s^e = (42954)^{19823} = 22968 \pmod{n}$. We have $s^e \neq m \pmod{n}$, i.e., the forged signature is not verified.

Security of RSA

If n can be factored, $\phi(n)$ can be computed and so d can be determined from e by extended gcd computation. Once d is known, any ciphertext can be decrypted and any signature can be forged.

At present no method (other than factoring n) is known to decrypt RSA-encrypted messages or forge RSA signatures.

We say that RSA derives its security from the intractability of the IFP.

If e, d, n are known, there exists a probabilistic polynomial-time algorithm to factor n. So RSA key inversion is as difficult as IFP. But RSA decryption or signature forging without the knowledge of d *may be* easier than factoring n.

In practice, we require $|n| \geqslant 1024$ with each of p, q having nearly half the size of n, in order to achieve a decent level of security.

Diffie-Hellman key exchange

Alice and Bob decide about a prime p and a primitive root g modulo p.

Alice generates a random $a \in \{2, 3, \ldots, p-2\}$ and sends $g^a \pmod{p}$ to Bob.

Bob generates a random $b \in \{2, 3, \ldots, p-2\}$ and sends $g^b \pmod{p}$ to Alice.

Alice computes $g^{ab} = (g^b)^a \pmod{p}$.

Bob computes $g^{ab} = (g^a)^b \pmod{p}$.

The quantity $g^{ab} \pmod{p}$ is the secret shared by Alice and Bob.

Example of Diffie-Hellman key exchange

Alice and Bob first take $p = 91573$, $g = 67$.

Alice generates $a = 39136$ and sends $g^a = 48745 \pmod{p}$ to Bob.

Bob generates $b = 8294$ and sends $g^b = 69167 \pmod{p}$ to Alice.

Alice computes $(69167)^{39136} = 71989 \pmod{p}$.

Bob computes $(48745)^{8294} = 71989 \pmod{p}$.

The secret shared by Alice and Bob is 71989.

Security of DH key exchange

An eavesdropper knows p, g, g^a, g^b and desires to compute $g^{ab} \pmod{p}$, that is, the eavesdropper has to solve the DHP.

If discrete logs can be computed in $\mathbb{Z}_p^*$, then a can be computed from g^a and one subsequently obtains $g^{ab} = (g^b)^a \pmod{p}$. So algorithms for solving the DLP can be used to break DH key exchange.

Breaking DH key exchange *may be* easier than solving the DLP.

At present no method other than computing discrete logs in $\mathbb{Z}_p^*$ is known to break DH key exchange.

Practically, we require p to be of size $\geqslant 1024$ bits. The security does not depend on the choice of the primitive element g. However, a and b must be sufficiently randomly chosen.

ElGamal encryption

Key generation

The recipient selects a random big prime p and a primitive root g modulo p, chooses a random $d \in \{2, 3, \ldots, p-2\}$, and computes $y = g^d \pmod{p}$.

Public key: (p, g, y).
Private key: (p, g, d).

Encryption

Input: Plaintext message $m \in \mathbb{Z}_p$ and the recipient's public key (p, g, y).
Output: Encrypted message (s, t).
Computation:

Generate a random integer $d' \in \{2, 3, \ldots, p-2\}$.
Compute $s = g^{d'} \pmod{p}$ and $t = my^{d'} \pmod{p}$.

Decryption

Input: Encrypted message (s, t) and the recipient's private key (p, g, d).
Output: The recovered plaintext message $m = ts^{-d} \pmod{p}$.

Correctness of ElGamal encryption

We have $s = g^{d'} \pmod{p}$ and $t = my^{d'} = m(g^d)^{d'} = mg^{dd'} \pmod{p}$. Therefore, $m = tg^{-dd'} = t(g^{d'})^{-d} = ts^{-d} \pmod{p}$.

Example of ElGamal encryption

Take $p = 91573$ and $g = 67$. The recipient chooses $d = 23632$ and so $y = (67)^{23632} = 87955 \pmod{p}$.

Let $m = 29485$ be the message to be encrypted. The sender chooses $d' = 1783$ and computes $s = g^{d'} = 52958 \pmod{p}$ and $t = my^{d'} = 1597 \pmod{p}$.

The recipient retrieves $m = ts^{-d} = 1597 \times (52958)^{-23632} = 29485 \pmod{p}$.

Security of ElGamal encryption

An eavesdropper knows g, p, y, s, t, where $y = g^d \pmod p$ and $s = g^{d'} \pmod p$. Determining m from (s, t) is equivalent to computing $g^{dd'} \pmod p$, since $t = mg^{dd'}$. (Here m is masked by the quantity $g^{dd'} \pmod p$.) But d, d' are unknown to the attacker. So the ability to solve the DHP lets the eavesdropper break ElGamal encryption.

Practically, we require p to be of size $\geqslant 1024$ bits for achieving a good level of security.

Part III

Public key cryptanalysis

Integer factoring algorithms

Let n be the integer to be factored.

Older algorithms

- Trial division (efficient if all prime divisors of n are small)
- Pollard's rho method
- Pollard's $p-1$ method (efficient if $p-1$ has only small prime factors for some prime divisor p of n)
- Williams' $p+1$ method (efficient if $p+1$ has only small prime factors for some prime divisor p of n)

In the worst case these algorithms are exponential in $\log n$.

Modern algorithms

Subexponential running time: $L(n, \omega, c) = \exp\left[(c + \mathrm{o}(1))(\ln n)^{\omega}(\ln \ln n)^{1-\omega}\right]$.

If $w = 0$, $L(n, \omega, c)$ is polynomial in $\ln n$.
If $w = 1$, $L(n, \omega, c)$ is exponential in $\ln n$.
For $0 < \omega < 1$, $L(n, \omega, c)$ is between polynomial and exponential.

Examples

Algorithm	Inventor(s)	Running time
Continued fraction method (CFRAC)	Morrison & Brillhart (1975)	$L(n, 1/2, c)$
Quadratic sieve method (QSM)	Pomerance (1984)	$L(n, 1/2, 1)$
Cubic sieve method (CSM)	Reyneri	$L(n, 1/2, 0.816)$
Elliptic curve method (ECM)	H W Lenstra (1987)	$L(n, 1/2, c)$
Number field sieve method (NFSM)	A K Lenstra, H W Lenstra, Manasse & Pollard (1990)	$L(n, 1/3, 1.923)$

Most modern integer factoring algorithms are based on Fermat's idea of writing a multiple of n as the difference of two squares.

Examples

Take $n = 899$. Then $n = 900 - 1 = 30^2 - 1^2 = (30 - 1) \times (30 + 1) = 29 \times 31$.

Then take $n = 833$. It is not immediate how we can write n as the difference of two squares. But $3 \times 833 = 2499 = 2500 - 1 = 50^2 - 1^2 = (50 - 1) \times (50 + 1) = 49 \times 51$. We have $\gcd(50 - 1, 833) = 49$, a non-trivial factor of 833.

Objective

We try to find integers $x, y \in \mathbb{Z}_n$ such that $x^2 = y^2 \pmod{n}$. Unless $x = \pm y \pmod{n}$, $\gcd(x - y, n)$ is a non-trivial divisor of n.

If n is composite, then for a randomly chosen pair (x, y) with $x^2 = y^2 \pmod{n}$, the probability that $x \neq \pm y \pmod{n}$ is at least $1/2$.

The quadratic sieve method (QSM)

Let n be an odd integer with no small prime factors. Let $H = \lceil \sqrt{n}\, \rceil$ and $J = H^2 - n$. For small integers $c \geqslant 0$ we have $(H+c)^2 = J + 2Hc + c^2 \pmod{n}$. We try to factor $T(c) = J + 2Hc + c^2$ completely over small primes $p_1, p_2, \ldots, p_t$. If the factorization is successful, we get a *relation*:

$$(H+c)^2 = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_t^{\alpha_t} \pmod{n}.$$

The left side is already a square. The right side is also a square if each α_i is even. But this is very rare. So we collect many relations:

$$\left.\begin{array}{lrcl} \text{Relation 1:} & (H+c_1)^2 & = & p_1^{\alpha_{11}} p_2^{\alpha_{12}} \cdots p_t^{\alpha_{1t}} \\ \text{Relation 2:} & (H+c_2)^2 & = & p_1^{\alpha_{21}} p_2^{\alpha_{22}} \cdots p_t^{\alpha_{2t}} \\ \quad\cdots & & & \\ \text{Relation } r\text{:} & (H+c_r)^2 & = & p_1^{\alpha_{r1}} p_2^{\alpha_{r2}} \cdots p_t^{\alpha_{rt}} \end{array}\right\} \pmod{n}.$$

Let $\beta_1, \beta_2, \ldots, \beta_r \in \{0, 1\}$. Consider the product of the β_i-th powers of Relations i.

$$\left[(H+c_1)^{\beta_1}(H+c_2)^{\beta_2}\cdots(H+c_r)^{\beta_r}\right]^2 = p_1^{\gamma_1}p_2^{\gamma_2}\cdots p_t^{\gamma_t} \pmod{n}.$$

Again the left side is a square. By tuning $\beta_1, \beta_2, \ldots, \beta_r$ we force each γ_i to be even. We have:

$$\begin{aligned}
\alpha_{11}\beta_1 + \alpha_{21}\beta_2 + \cdots + \alpha_{r1}\beta_r &= \gamma_1, \\
\alpha_{12}\beta_1 + \alpha_{22}\beta_2 + \cdots + \alpha_{r2}\beta_r &= \gamma_2, \\
&\cdots \\
\alpha_{1t}\beta_1 + \alpha_{2t}\beta_2 + \cdots + \alpha_{rt}\beta_r &= \gamma_t.
\end{aligned}$$

This is a system of t linear equations in r unknown quantities $\beta_1, \beta_2, \ldots, \beta_r$. Since each γ_i is required to be even, and since each $\gamma_i \in \{0, 1\}$, we solve the following system modulo 2:

$$\begin{pmatrix} \alpha_{11} & \alpha_{21} & \cdots & \alpha_{r1} \\ \alpha_{12} & \alpha_{22} & \cdots & \alpha_{r2} \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_{1t} & \alpha_{2t} & \cdots & \alpha_{rt} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_t \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \pmod 2.$$

For $r \geqslant t$, we hope to obtain non-zero solutions for $\beta_1, \beta_2, \ldots, \beta_r$. For each such solution we take

$$\begin{aligned} x &= (H + c_1)^{\beta_1}(H + c_2)^{\beta_2} \cdots (H + c_r)^{\beta_r} \pmod n, \\ y &= p_1^{\gamma_1/2} p_2^{\gamma_2/2} \cdots p_t^{\gamma_t/2} \pmod n. \end{aligned}$$

If $x \neq \pm y \pmod n$, then $\gcd(x - y, n)$ gives us a non-trivial factor of n.

Let p be a small prime. The condition $p \mid T(c)$ implies $(H + c)^2 = n \pmod p$. That is, if n is not a square modulo p, then p does not divide $T(c)$ for any value of c. So it suffices to consider only the small primes p modulo which n is a square.

Example of QSM

$n = 7116491.$

$H = \lceil \sqrt{n} \, \rceil = 2668.$

As the factor base we take all primes < 100 modulo which n is a square. This gives:

$B = \{2, 5, 7, 17, 29, 31, 41, 59, 61, 67, 71, 79, 97\}.$

$t = 13.$

We also take $r = 13$. In practice, one takes $r \approx 2t$.

The following relations are obtained.

$$
\left.\begin{array}{rrcl}
\text{Relation 1:} & (H+3)^2 & = & 2 \times 5^3 \times 71 \\
\text{Relation 2:} & (H+8)^2 & = & 5 \times 7 \times 31 \times 41 \\
\text{Relation 3:} & (H+49)^2 & = & 2 \times 41^2 \times 79 \\
\text{Relation 4:} & (H+64)^2 & = & 7 \times 29^2 \times 59 \\
\text{Relation 5:} & (H+81)^2 & = & 2 \times 5 \times 7^2 \times 29 \times 31 \\
\text{Relation 6:} & (H+109)^2 & = & 2 \times 7 \times 17 \times 41 \times 61 \\
\text{Relation 7:} & (H+128)^2 & = & 5^3 \times 71 \times 79 \\
\text{Relation 8:} & (H+145)^2 & = & 2 \times 71^2 \times 79 \\
\text{Relation 9:} & (H+182)^2 & = & 17^2 \times 59^2 \\
\text{Relation 10:} & (H+228)^2 & = & 5^2 \times 7^2 \times 17 \times 61 \\
\text{Relation 11:} & (H+267)^2 & = & 2 \times 7^2 \times 17 \times 29 \times 31 \\
\text{Relation 12:} & (H+382)^2 & = & 7 \times 59 \times 67 \times 79 \\
\text{Relation 13:} & (H+411)^2 & = & 2 \times 5^4 \times 31 \times 61
\end{array}\right\} \pmod{n}.
$$

We get the linear system:

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 & 3 & 0 & 0 & 2 & 0 & 0 & 4 \\ 0 & 1 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 2 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \\ \beta_9 \\ \beta_{10} \\ \beta_{11} \\ \beta_{12} \\ \beta_{13} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \pmod 2.$$

This system has 16 solutions. These solutions and the corresponding values of x, y and $\gcd(x-y, n)$ are given in the next slide. We obtain $n = 1847 \times 3853$.

β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9	β_{10}	β_{11}	β_{12}	β_{13}	x	y	$\gcd(x-y,n)$
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	7116491
1	0	1	0	0	0	1	0	0	0	0	0	0	1755331	560322	1847
0	0	1	0	0	0	0	1	0	0	0	0	0	526430	459938	1847
1	0	0	0	0	0	1	1	0	0	0	0	0	7045367	7045367	7116491
0	0	0	0	0	0	0	0	1	0	0	0	0	2850	1003	1847
1	0	1	0	0	0	1	0	1	0	0	0	0	6916668	6916668	7116491
0	0	1	0	0	0	0	1	1	0	0	0	0	5862390	5862390	7116491
1	0	0	0	0	0	1	1	1	0	0	0	0	3674839	6944029	1847
0	1	0	0	1	1	0	0	0	0	1	0	1	1079130	3965027	3853
1	1	1	0	1	1	1	0	0	0	1	0	1	5466596	1649895	1
0	1	1	0	1	1	0	1	0	0	1	0	1	5395334	1721157	1
1	1	0	0	1	1	1	1	0	0	1	0	1	6429806	3725000	3853
0	1	0	0	1	1	0	0	1	0	1	0	1	1196388	5920103	1
1	1	1	0	1	1	1	0	1	0	1	0	1	1799801	3818773	3853
0	1	1	0	1	1	0	1	1	0	1	0	1	5081340	4129649	3853
1	1	0	0	1	1	1	1	1	0	1	0	1	7099266	17225	1

Algorithms for computing discrete logarithms

Suppose that we want to compute the discrete logarithm of a in $\mathbb{Z}_p^*$ with respect to the primitive root g.

Older algorithms

- Brute-force search
- Shanks' Baby-step-giant-step method
- Pollard's rho method
- Pollard's lambda method
- Pohlig-Hellman method (Efficient if $p - 1$ has only small prime divisors)

In the worst case, these algorithms are exponential in $\log p$.

Modern algorithms

Based on the index calculus method (ICM)

Subexponential running time: $L(p, \omega, c) = \exp\left[(c + \mathrm{o}(1))(\ln p)^{\omega}(\ln \ln p)^{1-\omega}\right]$.

Examples

Algorithm	**Inventor(s)**	**Running time**
Basic ICM	Western & Miller (1968)	$L(p, 1/2, c)$
Linear sieve method (LSM) Residue list sieve method Gaussian integer method	Coppersmith, Odlyzko & Schroeppel (1986)	$L(p, 1/2, 1)$
Cubic sieve method (CSM)	Reyneri	$L(p, 1/2, 0.816)$
Number field sieve method (NFSM)	Gordon (1993)	$L(p, 1/3, 1.923)$

The basic index calculus method

Suppose we want to compute $\mathrm{ind}_g\, a$ in $\mathbb{Z}_p^*$.

Precomputation stage

Let $B = \{p_1, p_2, \ldots, p_t\}$ comprise the first t primes. B is called the *factor base*.

In this stage we compute the discrete logs $d_i = \mathrm{ind}_g\, p_i$ for $i = 1, 2, \ldots, t$.

Generate random $j \in \{1, 2, \ldots, p-2\}$ and compute $g^j \pmod{p}$. If this quantity factors completely over B, we get a *relation*:

$$g^j = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_t^{\alpha_t} \pmod{p}.$$

Taking discrete logarithm we get:

$$j = \alpha_1 d_1 + \alpha_2 d_2 + \cdots + \alpha_t d_t \pmod{p-1}.$$

This is a linear equation (modulo $p-1$) in t unknown quantities $d_1, d_2, \ldots, d_t$.

Generate $r \geqslant t$ relations for different values of j:

$$\left.\begin{array}{lrcl} \text{Relation 1:} & j_1 & = & \alpha_{11}d_1 + \alpha_{12}d_2 + \cdots + \alpha_{1t}d_t \\ \text{Relation 2:} & j_2 & = & \alpha_{21}d_1 + \alpha_{22}d_2 + \cdots + \alpha_{2t}d_t \\ \quad\cdots & & & \\ \text{Relation } r\text{:} & j_r & = & \alpha_{r1}d_1 + \alpha_{r2}d_2 + \cdots + \alpha_{rt}d_t \end{array}\right\} \pmod{p-1}.$$

Solve the system modulo $p-1$ to determine the unknown indices $d_1, d_2, \ldots, d_t$.

The second stage

Again we choose random j and try to factor $ag^j \pmod{p}$ completely over B. If the factorization is successful, we have:

$$ag^j = p_1^{\beta_1} p_2^{\beta_2} \cdots p_t^{\beta_t} \pmod{p}, \quad \text{i.e.,}$$

$$\operatorname{ind}_g a = -j + \beta_1 d_1 + \beta_2 d_2 + \cdots + \beta_t d_t \pmod{p-1}.$$

Substituting the values of $d_1, d_2, \ldots, d_t$ gives $\operatorname{ind}_g a$.

Example of the basic ICM

We take $p = 839$, $g = 31$, and $B = \{2, 3, 5, 7, 11\}$, i.e., $t = 5$. In order to obtain a matrix with rank t, we usually require $r \geqslant 2t$. So take $r = 10$.

The following relations are generated:

$$\left.\begin{array}{lrcl}
\text{Relation 1:} & g^{118} & = & 2^3 \times 5^2 \\
\text{Relation 2:} & g^{574} & = & 2^7 \times 5 \\
\text{Relation 3:} & g^{318} & = & 2^2 \times 3^3 \\
\text{Relation 4:} & g^{46} & = & 2^7 \\
\text{Relation 5:} & g^{786} & = & 2^2 \times 3^3 \times 7 \\
\text{Relation 6:} & g^{323} & = & 2 \times 3 \times 11 \\
\text{Relation 7:} & g^{606} & = & 3^4 \\
\text{Relation 8:} & g^{252} & = & 2^3 \times 3^2 \times 7 \\
\text{Relation 9:} & g^{160} & = & 3 \times 5^2 \\
\text{Relation 10:} & g^{600} & = & 2 \times 3^3 \times 5
\end{array}\right\} \pmod{p}.$$

This leads to the linear system given in the next slide.

$$\begin{pmatrix} 3 & 0 & 2 & 0 & 0 \\ 7 & 0 & 1 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 4 & 0 & 0 & 0 \\ 3 & 2 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 1 & 3 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{pmatrix} = \begin{pmatrix} 118 \\ 574 \\ 318 \\ 46 \\ 786 \\ 323 \\ 606 \\ 252 \\ 160 \\ 600 \end{pmatrix} \pmod{p-1}.$$

The coefficient matrix is of rank 5 modulo 838, and the system has the solution:

$$\left.\begin{array}{rcccl} d_1 & = & \operatorname{ind}_{31} 2 & = & 246 \\ d_2 & = & \operatorname{ind}_{31} 3 & = & 780 \\ d_3 & = & \operatorname{ind}_{31} 5 & = & 528 \\ d_4 & = & \operatorname{ind}_{31} 7 & = & 468 \\ d_5 & = & \operatorname{ind}_{31} 11 & = & 135 \end{array}\right\} \pmod{p-1}.$$

Let us now compute individual logarithms:

For $a = 561$, we have:

$$\begin{aligned} ag^{312} &= 600 = 2^3 \times 3 \times 5^2 \pmod{p}, \quad \textbf{i.e.,} \\ \operatorname{ind}_{31} 561 &= -312 + 3 \times 246 + 780 + 2 \times 528 = 586 \pmod{p-1}. \end{aligned}$$

For $a = 89$, we have:

$$\begin{aligned} ag^{342} &= 99 = 3^2 \times 11 \pmod{p}, \quad \textbf{i.e.,} \\ \operatorname{ind}_{31} 89 &= -342 + 2 \times 780 + 135 = 515 \pmod{p-1}. \end{aligned}$$

For $a = 625$, we have:

$$\begin{aligned} ag^{806} &= 70 = 2 \times 5 \times 7 \pmod{p}, \quad \textbf{i.e.,} \\ \operatorname{ind}_{31} 625 &= -806 + 246 + 528 + 468 = 436 \pmod{p-1}. \end{aligned}$$

Part IV

Stream ciphers

Stream ciphers encrypt data in small blocks, typically bit-by-bit.

Let $m = m_1 m_2 m_3 \ldots m_l$ be the plaintext message and $k = k_1 k_2 k_3 \ldots k_l$ a key-stream, where each m_i and each k_i are bits. The bits of the ciphertext $c = c_1 c_2 c_3 \ldots c_l$ are calculated as

$$c_i = m_i \oplus k_i \, ,$$

where $\oplus$ is the XOR operation. Decryption is done as

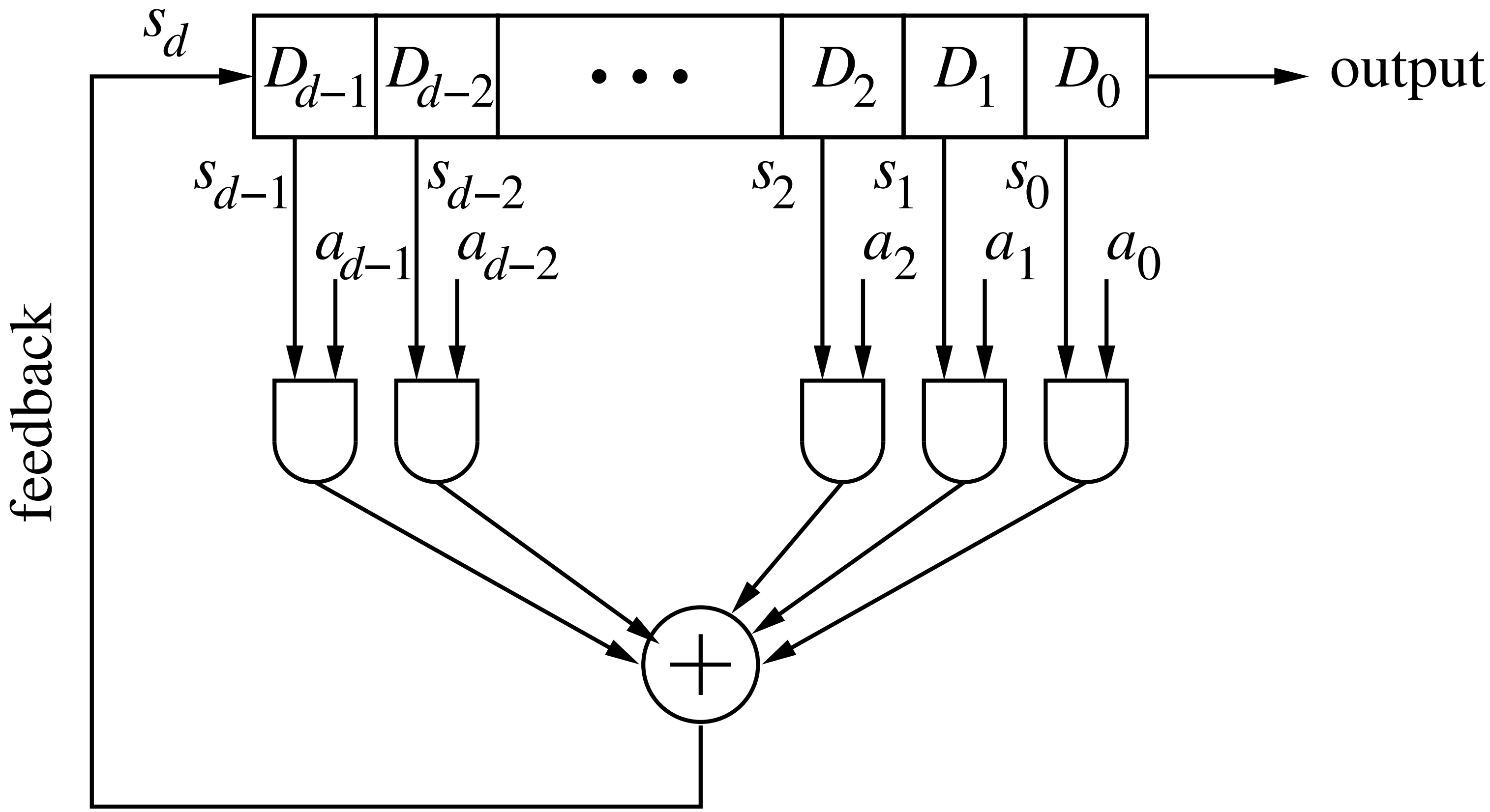
$$m_i = c_i \oplus k_i \, .$$

If the key stream is truly random, then $\Pr(c_i = 0) = \Pr(c_i = 1) = \frac{1}{2}$ for each i, irrespective of the probability of the values assumed by m_i. This cipher is called *one time pad* and is *unconditionally secure*.

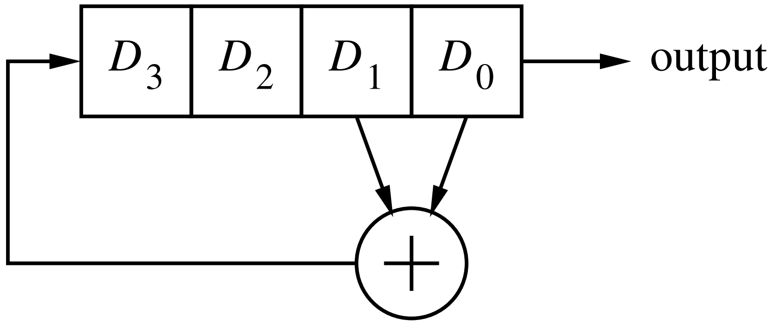
One time pad is not practical, since:

- The key length is as big as the message length.
- It is difficult to generate truly random (reproducible) bit streams.

Linear feedback shift register (LFSR)



Example



Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1
10	1	1	0	0
11	0	1	1	0
12	1	0	1	1
13	0	1	0	1
14	1	0	1	0
15	1	1	0	1

The bit values s_i stored in the delay elements D_i define the *state* of the LFSR. Let $(s_0, s_1, \ldots, s_{d-1})$ be the state of an LFSR at an instant. After a clock pulse, the state changes to $(t_0, t_1, \ldots, t_{d-1})$, where:

$$
\begin{aligned}
t_0 &= s_1 \\
t_1 &= s_2 \\
&\vdots \\
t_{d-2} &= s_{d-1} \\
t_{d-1} &= a_0 s_0 + a_1 s_1 + a_2 s_2 + \cdots + a_{d-1} s_{d-1} \pmod 2.
\end{aligned}
$$

Also the bit s_0 is output.

If the control bits $a_0, a_1, \ldots, a_{d-1}$ are suitably selected, then the bit stream output by the LFSR *appears* reasonably random.

The control bits $a_0, a_1, \ldots, a_{d-1}$ need to be kept secret. The initial state may additionally be kept secret.

Because of the linear relation of the feedback bit as a function of the current state, LFSRs are vulnerable to several attacks.

A known plaintext attack: Suppose that the bits m_i and c_i for $2d$ consecutive values of i (say, $1, 2, \ldots, 2d$) are known to an attacker. Then $k_i = m_i \oplus c_i$ are also known for these values of i. Define the states $S_i = (k_i, k_{i+1}, \ldots, k_{i+d-1})$ of the LFSR. We then have $S_{i+1} = \Delta S_i \pmod 2$ for $i = 1, 2, \ldots, d$, where Δ is the $d \times d$ transition matrix

$$\Delta = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ a_0 & a_1 & a_2 & \cdots & a_{d-2} & a_{d-1} \end{pmatrix}.$$

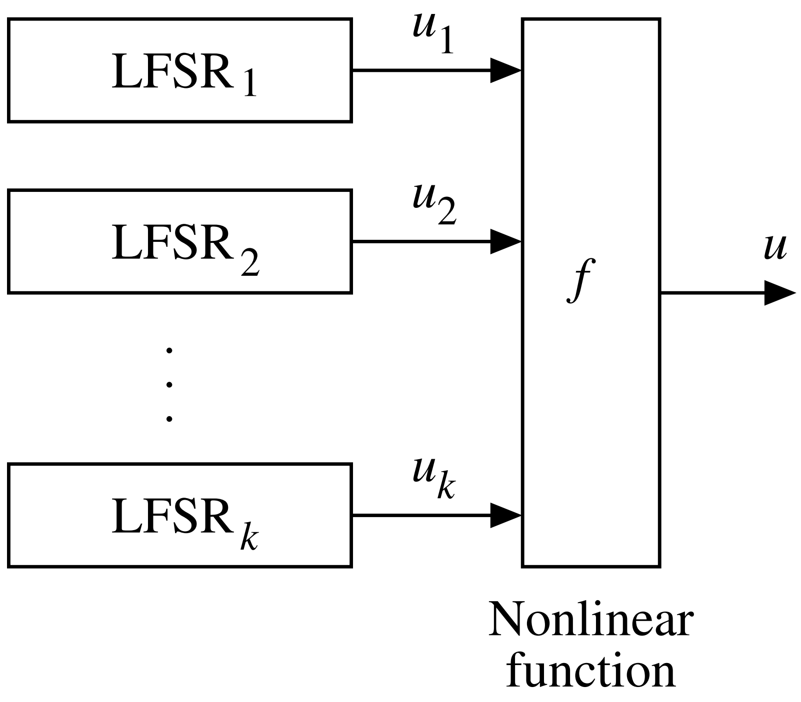
Treat each S_i as a column vector. We then have

$$\begin{pmatrix} S_2 & S_3 & \cdots & S_{d+1} \end{pmatrix} = \Delta \begin{pmatrix} S_1 & S_2 & \cdots & S_d \end{pmatrix} \pmod 2$$

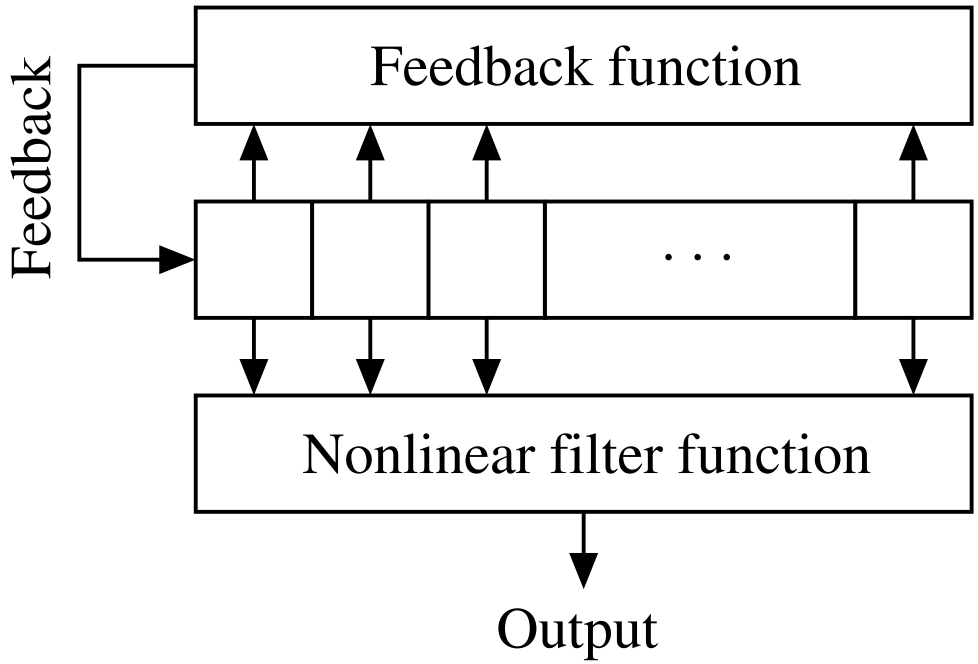
This reveals Δ and consequently the secret $a_0, a_1, \ldots, a_{d-1}$ with high probability.

Remedy: Introduce non-linearity in the LFSR outputs.

Nonlinear combination generator



Nonlinear filter generator



Part V

Hash functions

A hash function H maps a bit string of any length to a bit string of a fixed length n.

A *collision* for H is a pair (x, y) of bit strings with $H(x) = H(y)$. Collisions must exist for any hash function. For cryptographic purposes, collisions should be difficult to locate.

Desirable properties

Collision resistance: It should be computationally difficult to find two different bit strings x, y with $H(x) = H(y)$.

Second preimage resistance: Given a bit-string x, it should be difficult to compute another bit string y with $H(x) = H(y)$.

First preimage resistance: Given an n-bit string h, it should be difficult to compute a bit string x with $H(x) = h$. We exclude few values of h, since anybody can maintain $(x, H(x))$ for some strings x of one's choice.

Easy to compute

Theorem: Collision resistance implies second preimage resistance.

Second preimage resistance does not imply collision resistance: Let S be a finite set of size $\geqslant 2$ and let H be a cryptographic hash function. Then

$$H'(x) = \begin{cases} 0^{n+1} & \text{if } x \in S, \\ 1 \,||\, H(x) & \text{otherwise,} \end{cases}$$

is second preimage resistant but not collision resistant.

Collision resistance does not imply first preimage resistance: Let H be an n-bit cryptographic hash function. Then

$$H''(x) = \begin{cases} 0 \,||\, x & \text{if } |x| = n, \\ 1 \,||\, H(x) & \text{otherwise.} \end{cases}$$

is collision resistant (so second preimage resistant), but not first preimage resistant.

First preimage resistance does not imply second preimage resistance: Let m be a product of two unknown big primes. Define $H'''(x) = (1 \,||\, x)^2 \pmod{m}$. H''' is first preimage resistant, but not second preimage resistant.

Merkle-Damgård construction

A *compression function* maps bit strings of length m to bit strings of length n, where $m = n + r$ for some $r \geqslant 1$.

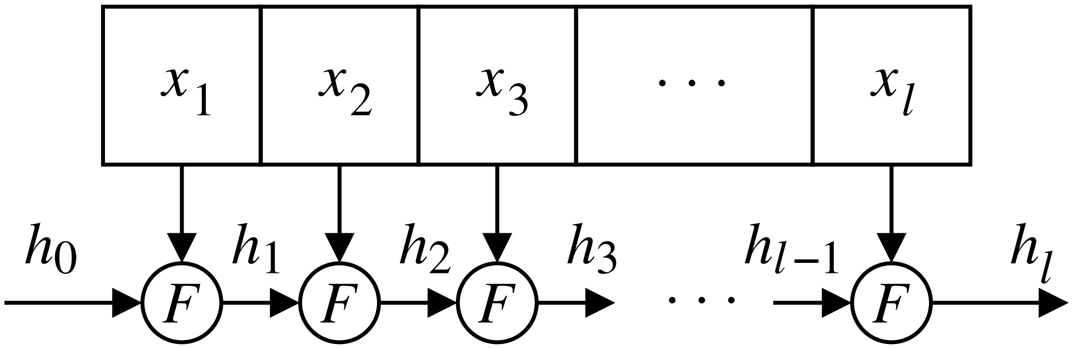
Given a compression function F, we construct a hash function H.

Break the input message $x = x_1 \,||\, x_2 \,||\, \cdots \,||\, x_l$ into blocks x_i each of length r bits.

Initialize $h_0 = 0^n$.

For $i = 1, 2, \ldots, l$ compute $h_i = F(h_{i-1} \,||\, x_i)$.

Output $H(x) = h_l$.



Theorem: If F is collision resistant, then H is also collision resistant.

Theorem: If F is first preimage resistant, then H is also first preimage resistant.

Example of a compression function: $F(M \,||\, K) = E(M, K)$, where E is a block cipher, K is the key and M is the plaintext message.

Custom-designed hash functions

The MD (Message Digest) family: MD2, MD4, and MD5 produce 128-bit hash values. MD4 was broken soon after its conception.

RIPEMD: 160-bit hash values.

The SHA (Secure Hash Algorithm) family: SHA-1, SHA-256, SHA-384 and SHA-512 respectively produce 160-, 256-, 384- and 512-bit hash values.

Attacks on hash functions

Birthday attack: For an n-bit hash function, $O(2^{n/2})$ hash values need to be computed in order to detect a collision with high probability. In view of this $n \geqslant 160$ is recommended.

Attacks on the underlying compression function

Attacks on the underlying block cipher

Chaining attacks

Keyed hash functions

Used to authenticate the source of a message and to preserve data integrity.

$\mathrm{HMAC}(M) = H(K \,||\, P \,||\, H(K \,||\, Q \,||\, M))$, where H is an unkeyed hash function, K is a key and P, Q are short padding strings.