Introduction to Cryptography

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

Part I

Overview of cryptographic primitives

What is Cryptography?

Cryptography is the study of techniques for preventing access to sensitive data by parties who are not authorized to access the data.

Cryptanalysis is the study of techniques for breaking cryptographic systems.

Cryptology = Cryptography + Cryptanalysis

Cryptanalysis is useful for strengthening cryptographic primitives.

Maintaining security and privacy is an ancient and primitive need.

Particularly relevant for military and diplomatic applications.

Wide deployment of the Internet makes everybody a user of cryptographic tools.

Message encryption

Required for secure transmission of messages over a public channel.

Alice wants to send a **plaintext** message M to Bob.

Alice **encrypts** M to generate the **ciphertext** message $C = f_e(M, K_e)$.

K_e is the **encryption key**.

C is sent to Bob over the public channel.

Bob **decrypts** C to recover the plaintext message $M = f_d(C, K_d)$.

K_d is the **decryption key**.

Knowledge of K_d is required to retrieve M from C.

An eavesdropper (intruder, attacker, adversary, opponent, enemy) cannot decrypt C.

Secret-key or symmetric encryption

$K_e = K_d$.

Algorithms are fast and suitable for software and hardware implementations.

The common key has to be agreed upon by Alice and Bob before the actual communication.

Each pair of communicating parties needs a secret key.

If there are many communicating pairs, the key storage requirement is high.

Public-key or asymmetric encryption

$K_e \neq K_d$.

Introduced by Rivest, Shamir and Adleman (1978).

K_e is the **public key** known to everybody (even to enemies).

K_d is the **private key** to be kept secret.

It is difficult to compute K_d from K_e.

Anybody can send messages to anybody. Only the proper recipient can decrypt.

No need to establish keys a priori.

Each party requires only one key-pair for communicating with everybody.

Algorithms are slow, in general.

Real-life analogy

Symmetric encryption

Alice locks the message in a box by a key.

Bob uses a copy of the same key to unlock.

Asymmetric encryption

Alice presses a padlock in order to lock the box. The locking process does not require a real key.

Bob has the key to open the padlock.

Using symmetric and asymmetric encryption together

Alice reads Bob's public key K_e.

Alice generates a random secret key K.

Alice encrypts M by K to generate $C = f_e(M, K)$.

Alice encrypts K by K_e to generate $L = f_E(K, K_e)$.

Alice sends (C, L) to Bob.

Bob recovers K as $K = f_D(L, K_d)$.

Bob decrypts C as $M = f_d(C, K)$.

Key agreement or key exchange

Real-life analogy

Alice procures a lock L with key K. Alice wants to send K to Bob for a future secret communication.

Alice procures another lock L_A with key K_A to be used at Alice's end only.

Bob procures a lock L_B with key K_B to be used at Bob's end only.

Alice puts K in a box, locks the box by L_A using K_A, and sends the box to Bob.

Bob locks the box by L_B using K_B, and sends the doubly-locked box back to Alice.

Alice unlocks L_A by K_A and sends the box again to Bob.

Bob unlocks L_B by K_B and obtains K.

A third party always finds the box locked either by L_A or L_B or both.

Key agreement or key exchange (contd)

Alice generates a key pair (A_e, A_d).

Bob generates a key pair (B_e, B_d).

Alice sends her public-key A_e to Bob.

Bob sends his public-key B_e to Alice.

Alice computes $K_{AB} = f(A_e, A_d, B_e)$.

Bob computes $K_{BA} = f(B_e, B_d, A_e)$.

The protocol insures $K_{AB} = K_{BA}$ to be used by Alice and Bob as a shared secret.

An intruder cannot compute this secret using A_e and B_e only.

Digital signatures

Alice establishes her binding to a message M by digitally signing it.

Signing: Only Alice has the capability to sign M.

Verification: Anybody can verify whether Alice's signature on M is valid.

Forging: Nobody can forge signatures on behalf of Alice.

Digital signatures are based on public-key techniques.

Signature generation $\equiv$ Decryption (uses private key), and
Signature verification $\equiv$ Encryption (uses public key).

Non-repudiation: An entity should not be allowed to deny valid signatures made by him.

Signature with message recovery

Generation

Alice generates a key-pair (K_e, K_d), publishes K_e, and keeps K_d secret.

Alice signs M by her private key to obtain the signed message $S = f_s(M, K_d)$.

Verification

Anybody can recover M from S by using Alice's public key: $M = f_v(S, K_e)$.

Forging signatures

A key K'_d other than K_d is used to generate the forged signature $S' = f_s(M, K'_d)$. Verification yields $M' = f_v(S', K_e)$. We would have $M' \neq M$. M' is not expected to have the same redundancy as M has, and so S' is rejected.

Drawback

Public-key algorithms are slow. This is of concern for signing long messages.

Signature with appendix

Generation

Alice generates a key-pair (K_e, K_d), publishes K_e, and keeps K_d secret.

Alice generates a short representative $m = H(M)$ of M.

Alice uses her private-key: $s = f_s(m, K_d)$.

Alice publishes (M, s) as the signed message.

Verification

Compute the representative $m = H(M)$.

Use Alice's public-key to generate $m' = f_v(s, K_e)$.

Accept the signature if and only if $m = m'$.

Forging

Verification is expected to fail if a key $K'_d \neq K_d$ is used to generate s.

Digital signatures: classification

Deterministic signatures: For a given message the same signature is generated on every occasion the signing algorithm is executed.

Probabilistic signatures: On different runs of the signing algorithm different signatures are generated, even if the message remains the same.

Probabilistic signatures offer better protection against some kinds of forgery.

Deterministic signatures are of two types:

- **Multiple-use signatures:** Slow. Parameters are used multiple times.
- **One-time signatures:** Fast. Parameters are used only once.

Entity authentication

Alice proves her identity to Bob.

Alice demonstrates to Bob her knowledge of a secret piece of information.

Alice may or may not reveal the secret itself to Bob.

Both symmetric and asymmetric techniques are used for entity authentication.

Weak authentication: Passwords

Set-up phase

Alice supplies a secret password P to Bob.

Bob transforms (typically encrypts) P to generate $Q = f(P)$.

Bob stores Q for future use.

Authentication phase

Alice supplies her password P' to Bob.

Bob computes $Q' = f(P')$.

Bob compares Q' with the stored value Q.

$Q' = Q$ if and only if $P' = P$.

If $Q' = Q$, Bob accepts Alice's identity.

Passwords (contd)

It should be difficult to invert the initial transform $Q = f(P)$.

Knowledge of Q, even if readable by enemies, does not reveal P.

Drawbacks

Alice reveals P itself to Bob. Bob may misuse this information.

P resides in unencrypted form in the memory during the authentication phase. A third party having access to this memory obtains Alice's secret.

Challenge-response techniques

Alice does not reveal her secret directly to Bob.

Bob generates a challenge C and sends C to Alice.

Alice responds to C by sending a response R back to Bob.

Bob determines whether the response R is satisfactory.

Generating R from C requires the knowledge of the secret.

Absence of the knowledge of the secret fails to generate a satisfactory response with a significantly positive probability p.

The above protocol may be repeated more than once (depending on p).

If Bob receives satisfactory response in every iteration, he accepts Alice's identity.

Drawback

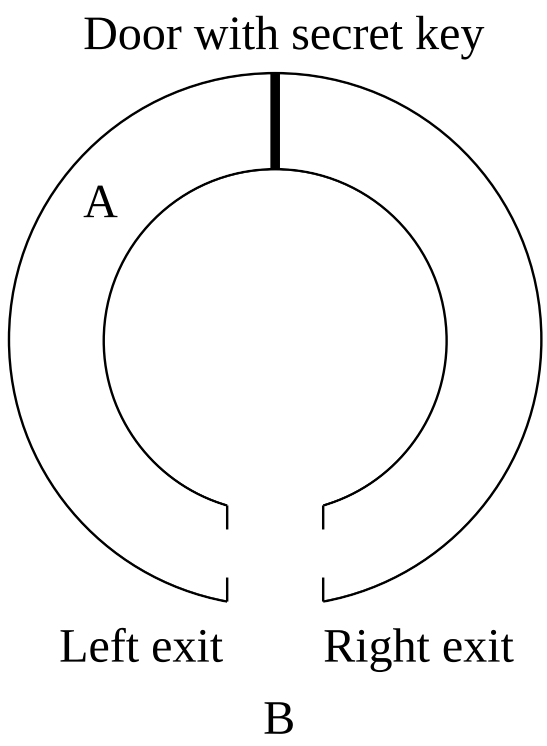
C and R may reveal to Bob or an eavesdropper some knowledge about Alice's secret.

Zero-knowledge protocol

A special class of challenge-response techniques.

Absolutely no information is leaked to Bob or to any third party.

A real-life example



Secret sharing

A secret is distributed to n parties.

All of these n parties should cooperate to reconstruct the secret.

Participation of only $\leqslant n-1$ parties should fail to reconstruct the secret.

Generalization

Any m (or more) parties can reconstruct the secret (for some $m \leqslant n$).

Participation of only $\leqslant m-1$ parties should fail to reconstruct the secret.

Cryptographic hash functions

Used to convert strings of any length to strings of a fixed length.

Used for the generation of (short) representatives of messages.

Symmetric techniques are typically used for designing hash functions.

Modification detection code (MDC)

An unkeyed hash function is used to guard against unauthorized/accidental message alterations. Signature schemes also use MDC's.

Message authentication code (MAC)

A keyed hash function is used to authenticate the source of messages.

Cryptographic hash functions: Properties

A **collision** for a hash function H is a pair of two distinct strings x, y with $H(x) = H(y)$. Collisions must exist for any hash function.

First pre-image resistance

For most hash values y, it should be difficult to find a string x with $H(x) = y$.

Second pre-image resistance

Given a string x, it should be difficult to find a different string x' with $H(x') = H(x)$.

Collision resistance

It should be difficult to find two distinct strings x, x' with $H(x) = H(x')$.

Certification

A **public-key certificate** insures that a public key actually belongs to an entity.

Certificates are issued by a trusted **Certification Authority** (CA).

A certificate consists of a public key and other additional information about the owner of the key.

The authenticity of a certificate is achieved by the digital signature of the CA on the certificate.

Compromised certificates are revoked and a **certificate revocation list** (CRL) is maintained by the CA.

If a certificate is not in the CRL, and the signature of the CA on the certificate is verified, one gains the desired confidence of treating the public-key as authentic.

Models of attack

Partial breaking of a cryptosystem

The attacker succeeds in decrypting some ciphertext messages, but without any guarantee that this capability would help him break new ciphertext messages in future.

Complete breaking of a cryptosystem

The attacker possesses the capability of decrypting any ciphertext message. This may be attributed to a knowledge of the decryption key(s).

Passive attack

The attacker only intercepts messages meant for others.

Active attack

The attacker alters and/or deletes messages and even creates unauthorized messages.

Types of passive attack

Ciphertext-only attack: The attacker has no control/knowledge of the ciphertexts and the corresponding plaintexts. This is the most difficult (but practical) attack.

Known plaintext attack: The attacker knows some plaintext-ciphertext pairs. Easily mountable in public-key systems.

Chosen plaintext attack: A known plaintext attack where the plaintext messages are chosen by the attacker.

Adaptive chosen plaintext attack: A chosen plaintext attack where the plaintext messages are chosen adaptively by the attacker.

Chosen ciphertext attack: A known plaintext attack where the ciphertext messages are chosen by the attacker. Mountable if the attacker gets hold of the victim's decryption device.

Adaptive chosen ciphertext attack: A chosen ciphertext attack where the ciphertext messages are chosen adaptively by the attacker.

Attacks on digital signatures

Total break: An attacker knows the signing key or has a function that is equivalent to the signature generation transformation.

Selective forgery: An attacker can generate signatures (without the participation of the legitimate signer) on a set of messages chosen by the attacker.

Existential forgery: The attacker can generate signatures on certain messages over which the attacker has no control.

Attacks on digital signatures (contd)

Key-only attack: The attacker knows only the verification (public) key of the signer. This is the most difficult attack to mount.

Known message attack: The attacker knows some messages and the signatures of the signer on these messages.

Chosen message attack: This is similar to the known message attack except that the messages for which the signatures are known are chosen by the attacker.

Adaptive chosen message attack: The messages to be signed are adaptively chosen by the attacker.

Part II

Symmetric cryptosystems

Block ciphers

A block cipher f of **block-size** n and **key-size** r is a function

$$f : \mathbb{Z}_2^n \times \mathbb{Z}_2^r \to \mathbb{Z}_2^n$$

that maps (M, K) to $C = f(M, K)$.

For each key K the map

$$f_K : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$$

taking a plaintext message M to the ciphertext message $C = f_K(M) = f(M, K)$ should be bijective (invertible).

n and r should be large enough to preclude successful exhaustive search.

Each f_K should be a sufficiently random permutation.

Block ciphers: Examples

Name	n	r
DES (Data Encryption Standard)	64	56
FEAL (Fast Data Encipherment Algorithm)	64	64
SAFER (Secure And Fast Encryption Routine)	64	64
IDEA (International Data Encryption Algorithm)	64	128
Blowfish	64	$\leqslant 448$
Rijndael	128/192/256	128/192/256

Old standard: DES

New standard: AES (adaptation of the Rijndael cipher)

A case study: AES (Advanced Encryption Standard)

AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.

Number of **rounds** N_r for AES is 10/12/14 for key-sizes 128/192/256.

AES **key schedule**: From K generate round keys $K_0, K_1, \ldots, K_{4N_r+3}$.

State: AES represents a 128-bit message block as a 4×4 array of octets:

$$\mu_0\mu_1\ldots\mu_{15} \equiv \begin{array}{|c|c|c|c|} \hline \mu_0 & \mu_4 & \mu_8 & \mu_{12} \\ \hline \mu_1 & \mu_5 & \mu_9 & \mu_{13} \\ \hline \mu_2 & \mu_6 & \mu_{10} & \mu_{14} \\ \hline \mu_3 & \mu_7 & \mu_{11} & \mu_{15} \\ \hline \end{array}$$

Each octet in the state is identified as an element of $\mathbb{F}_{2^8} = \mathbb{F}_2[x]/\langle x^8+x^4+x^3+x+1\rangle$.

Each column in the state is identified as an element of $\mathbb{F}_{2^8}[y]/\langle y^4+1\rangle$.

AES encryption

Generate the key schedule $K_0, K_1, \ldots, K_{4N_r+3}$ from the key K.

Convert the plaintext block M to a state S.

$S = \mathrm{AddKey}(S, K_0, K_1, K_2, K_3)$. [bitwise XOR]

for $i = 1, 2, \ldots, N_r$ do the following:

- $S = \mathrm{SubState}(S)$. [a non-linear transformation involving inverses in $\mathbb{F}_{2^8}$]
- $S = \mathrm{ShiftRows}(S)$. [cyclic shift of octets in each row]
- If $i \neq N_r$, $S = \mathrm{MixCols}(S)$. [a column-wise operation in $\mathbb{F}_{2^8}[y]/\langle y^4 + 1\rangle$]
- $S = \mathrm{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$. [bitwise XOR]

Convert the state S to the ciphertext block C.

AES decryption

Generate the key schedule $K_0, K_1, \ldots, K_{4N_r+3}$ from the key K.

Convert the ciphertext block C to a state S.

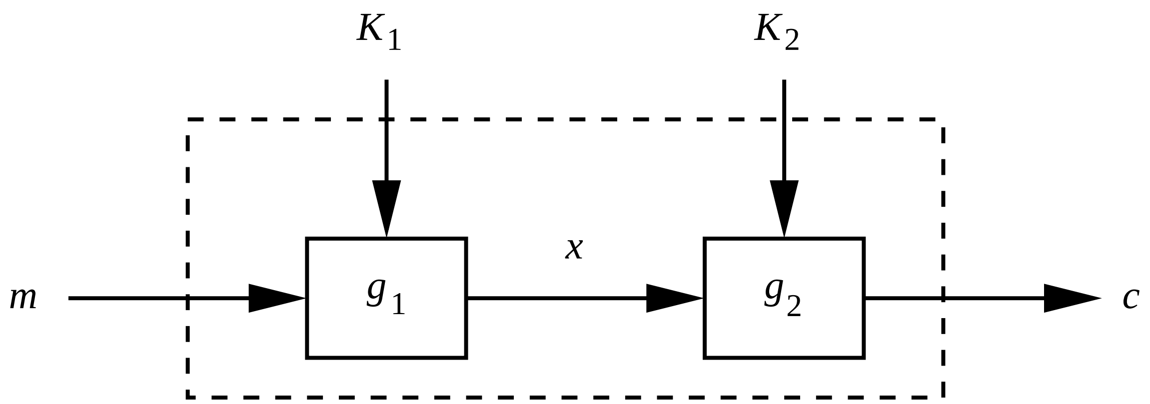
$S = \mathrm{AddKey}(S, K_{4N_r}, K_{4N_r+1}, K_{4N_r+2}, K_{4N_r+3})$.

for $i = N_r - 1, N_r - 2, \ldots, 1, 0$ do the following:

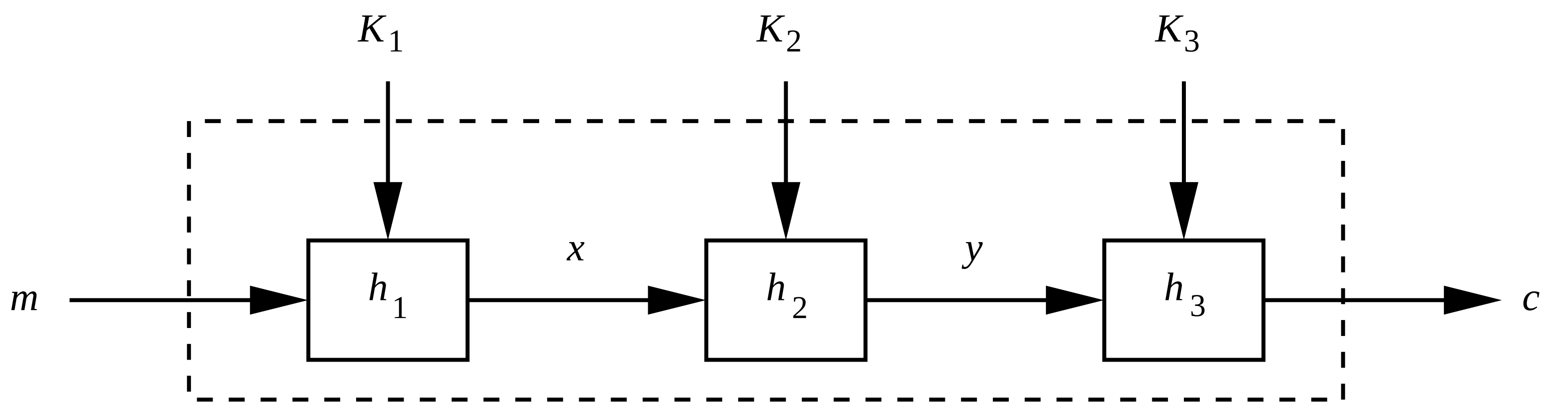
> $S = \mathrm{ShiftRows}^{-1}(S)$.
> $S = \mathrm{SubState}^{-1}(S)$.
> $S = \mathrm{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$.
> If $i \neq 0$, $S = \mathrm{MixCols}^{-1}(S)$.

Convert the state S to the plaintext block M.

Multiple encryption



(a) Double encryption



(b) Triple encryption

Modes of operation

Break the message $M = M_1 M_2 \ldots M_l$ into blocks each of bit-length $n' \leqslant n$.

ECB (Electronic Code-Book) mode: Here $n' = n$.

$C_i = f_K(M_i)$.

CBC (Cipher-Block Chaining) mode: Here $n' = n$.

$C_i = f_K(M_i \oplus C_{i-1})$.

CFB (Cipher FeedBack) Mode: Here $n' \leqslant n$. Initialize $k_0 = \mathrm{IV}$.

$C_i = M_i \oplus \mathrm{msb}_{n'}(f_K(k_{i-1}))$. [Mask the key and the plaintext block]

$k_i = \mathrm{lsb}_{n-n'}(k_{i-1}) \,||\, C_i$. [Generate the next key in the stream]

OFB (Output FeedBack) Mode: Here $n' \leqslant n$. Initialize $k_0 = \mathrm{IV}$.

$k_i = f_K(k_{i-1})$. [Generate the next key in the stream]

$C_i = M_i \oplus \mathrm{msb}_{n'}(k_i)$. [Mask the plaintext block]

Attacks on block ciphers

Exhaustive key search: If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs. Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits). Only two plaintext-ciphertext pairs usually suffice to determine a key uniquely.

Linear and differential cryptanalysis: By far the most sophisticated attacks on block ciphers. Impractical if sufficiently many rounds are used. AES is robust against these attacks.

Specific attacks on AES:

Square attack
Collision attack
Algebraic attacks (like XSL)

Meet-in-the-middle attack: Applies to multiple encryption schemes. With m stages we get the equivalent security of $\lceil m/2 \rceil$ keys only.

Stream ciphers

Stream ciphers encrypt bit-by-bit.

Plaintext stream: $M = m_1 m_2 \ldots m_l$.
Key stream: $K = k_1 k_2 \ldots k_l$.
Ciphertext stream: $C = c_1 c_2 \ldots c_l$.

Encryption: $c_i = m_i \oplus k_i$.

Decryption: $m_i = c_i \oplus k_i$.

Source of security: unpredictability in the key-stream.

Vernam's one-time pad: 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 probabilities of the values assumed by m_i. This leads to **unconditional security**, i.e., the knowledge of any number of plaintext-ciphertext bit pairs, does not help in decrypting a new ciphertext bit.

Stream ciphers: drawbacks

Key stream should be as long as the message stream. Management of long key streams is difficult.

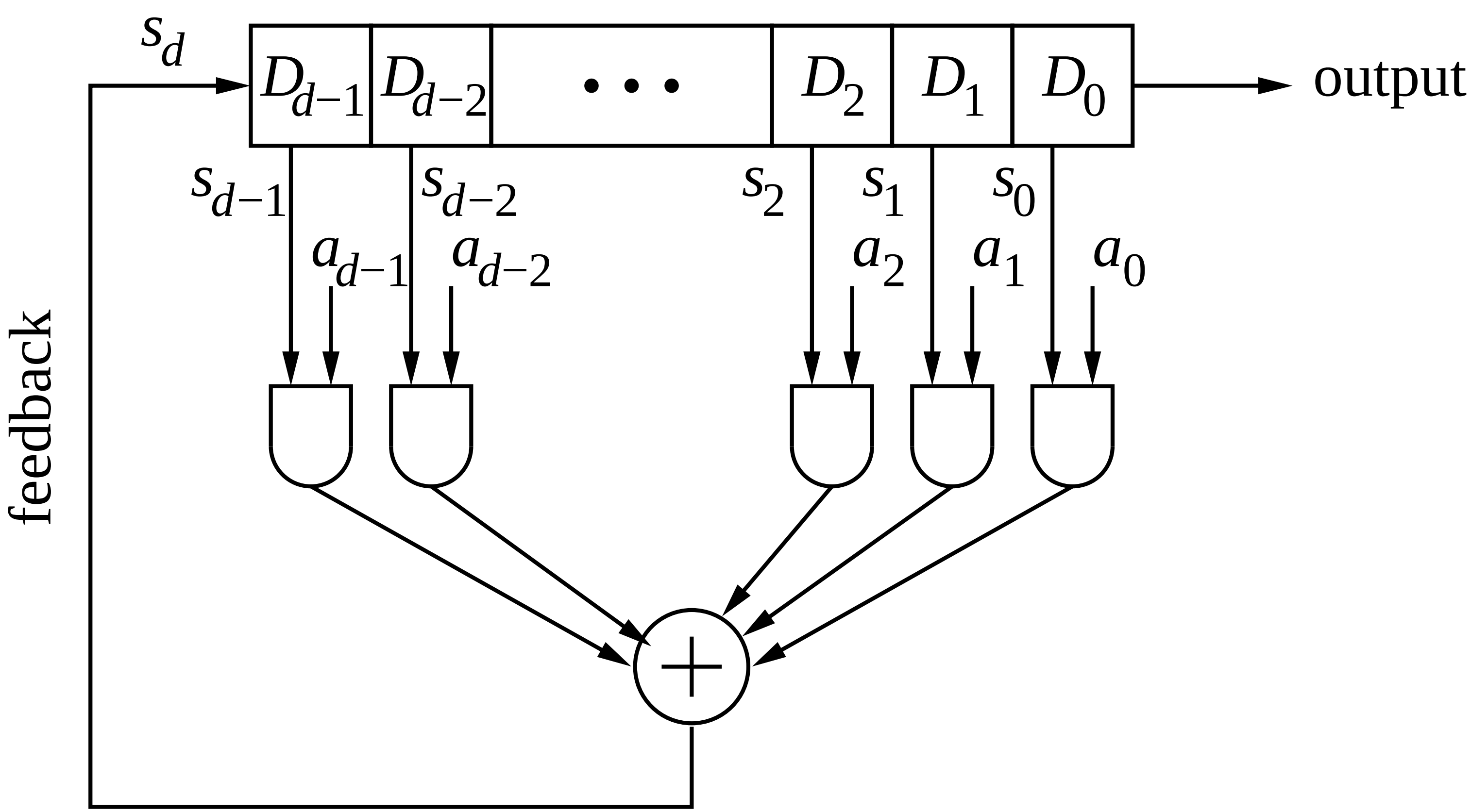
It is difficult to generate truly random (and reproducible) key streams.

Pseudorandom bit streams provide practical solution, but do not guarantee unconditional security.

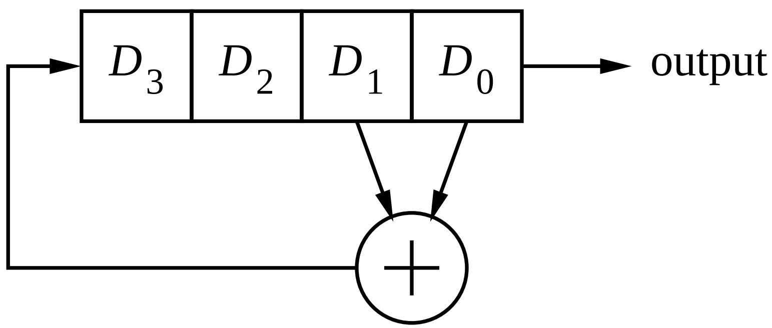
Pseudorandom bit generators are vulnerable to compromise of seeds.

Repeated use of the same key stream degrades security.

Linear Feedback Shift Registers (LFSR)



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

LFSR: State transition

Control bits: $a_0, a_1, \ldots, a_{d-1}$.

State: $\mathbf{s} = (s_0, s_1, \ldots, s_{d-1})$.

Each clock pulse changes the state as follows:

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

In the matrix notation $\mathbf{t} = \Delta_L \mathbf{s} \pmod 2$, where the **transition matrix** is

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

LFSR (contd)

The output bit-stream behaves like a pseudorandom sequence.

The output stream must be periodic. The period should be large.

Maximum period of a non-zero bit-stream = $2^d - 1$.

Maximum-length LFSR has the maximum period.

Connection polynomial

$$C_L(x) = 1 + a_{d-1}x + a_{d-2}x^2 + \cdots + a_1 x^{d-1} + a_0 x^d \in \mathbb{F}_2[X].$$

L is a maximum-length LFSR if and only if $C_L(x)$ is a primitive polynomial of $\mathbb{F}_2[x]$.

An attack on LFSR

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

Berlekamp-Massey 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

$$S_{i+1} = \Delta_L S_i \pmod{2}$$

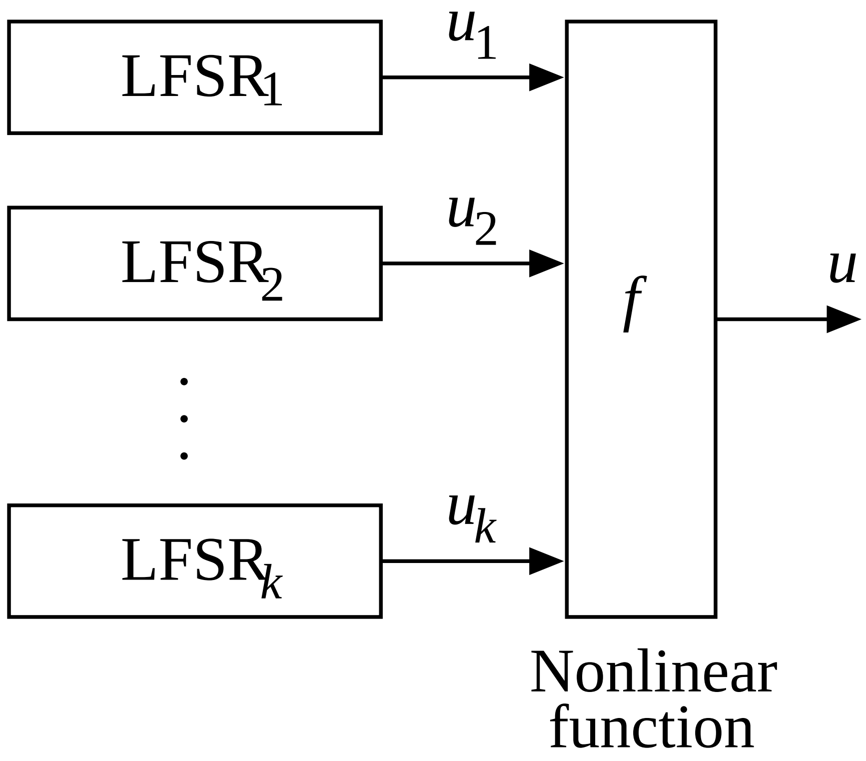
for $i = 1, 2, \ldots, d$. Treat each S_i as a column vector. We then have

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

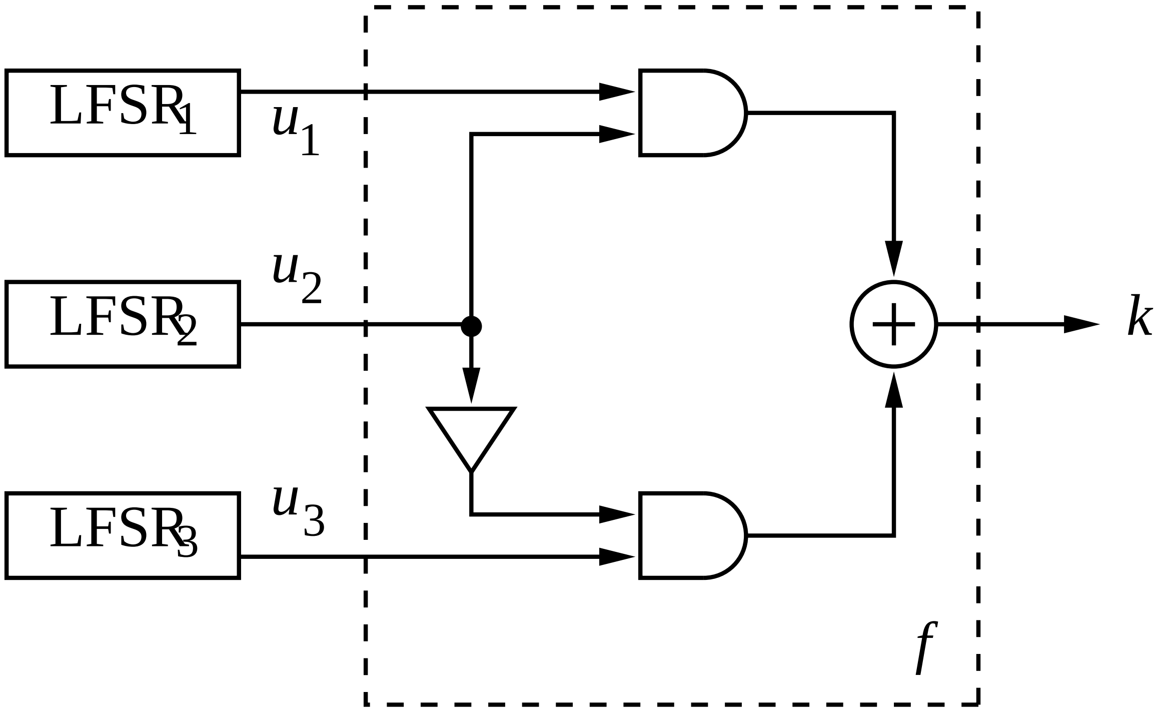
This reveals Δ_L and consequently the secret $a_0, a_1, \ldots, a_{d-1}$ with high probability.

Remedy: Introduce non-linearity to the LFSR output.

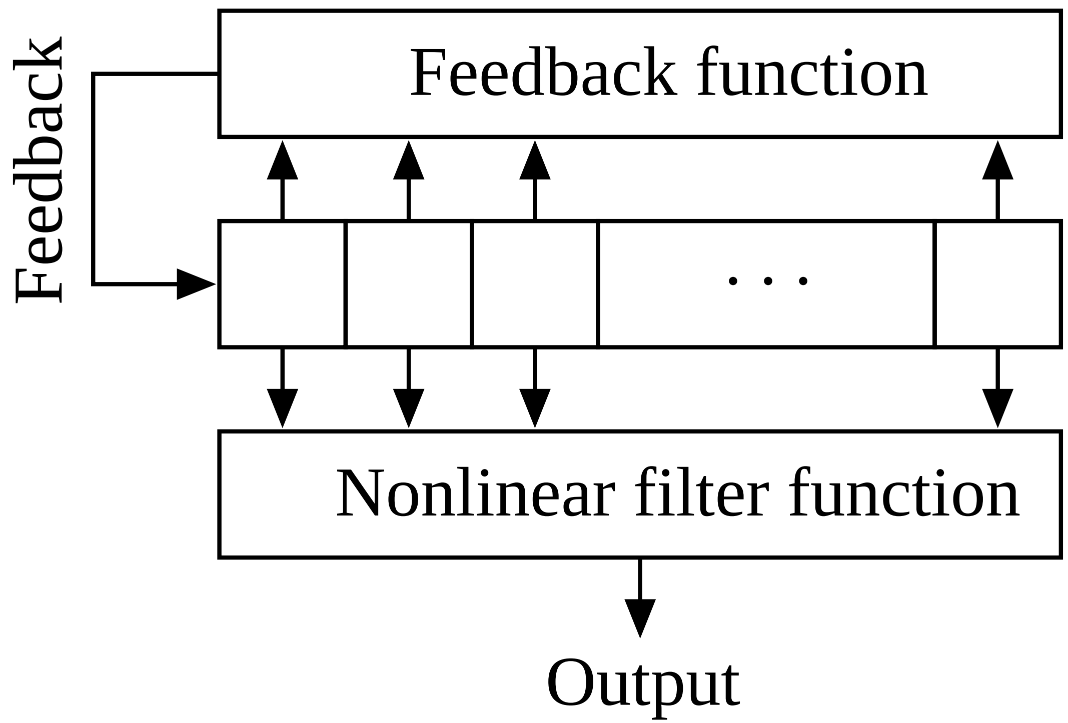
Nonlinear combination generator



The Geffe generator



Nonlinear filter generator



Hash functions

Collision resistance implies second pre-image resistance.

Second pre-image 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 pre-image resistant but not collision resistant.

Collision resistance does not imply first pre-image 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 pre-image resistant), but not first pre-image resistant.

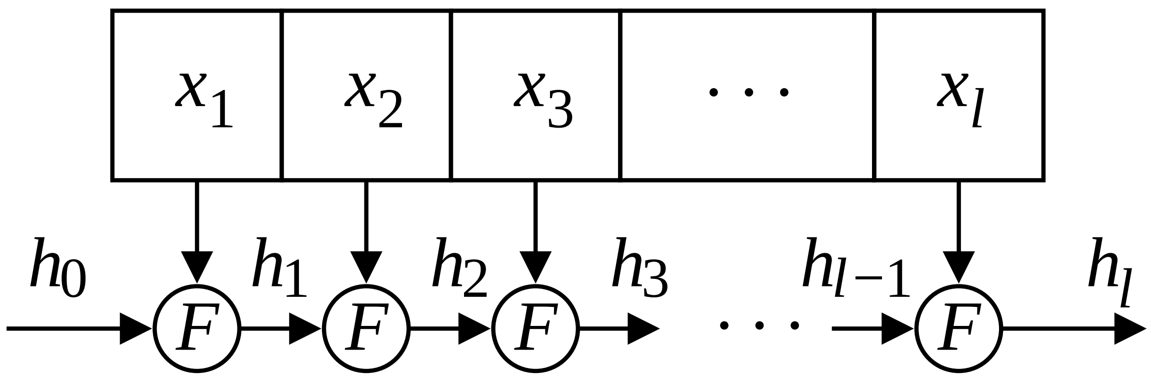
First pre-image resistance does not imply second pre-image resistance: Let m be a product of two unknown big primes. Define $H'''(x) = (1 \,||\, x)^2 \pmod{m}$. H''' is first pre-image resistant, but not second pre-image resistant.

Hash functions: Construction

Compression function: A function $F : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$, where $m = n + r$.

Merkle-Damgård's meta method

Break the input $x = x_1 x_2 \ldots x_l$ to blocks each of bit-length r.
Initialize $h_0 = 0^r$.
For $i = 1, 2, \ldots, l$ use compression $h_i = F(h_{i-1} \mid\mid x_i)$.
Output $H(x) = h_l$ as the hash value.



Hash functions: Construction (contd)

Properties

If F is first pre-image resistant, then H is also first pre-image resistant.

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

A concrete realization

Let f is a block cipher of block-size n and key-size r. Take:

$$F(M \,||\, K) = f_K(M).$$

Keyed hash function

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

Custom-designed hash functions

The SHA (Secure Hash Algorithm) family:

SHA-1 (160-bit), SHA-256 (256-bit), SHA-384 (384-bit), SHA-512 (512-bit).

The MD family:

MD2 (128-bit), MD5 (128-bit).

The RIPEMD family:

RIPEMD-128 (128-bit), RIPEMD-160 (160-bit).

Attacks on hash functions

The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.

For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).

Algebraic attacks may make hash functions vulnerable.

Some other attacks:

- Pseudo-collision attacks
- Chaining attacks
- Attacks on the underlying cipher
- Exhaustive key search for keyed hash functions
- Long message attacks

Part III

Public-key cryptosystems

Intractable problems

Public-key cryptography is based on **trapdoor one-way functions**. It should be easy to encrypt a message or verify a signature, but inverting the transform (decryption or signature generation) should be difficult, unless some secret information (the trapdoor) is known.

Several difficult computational problems are used to build the trapdoors. Examples:

- Factoring composite integers
- Computing square roots modulo a composite integer
- Computing discrete logarithms in certain groups (finite fields, elliptic and hyperelliptic curves, class group of number fields, etc.)
- Finding shortest/closest vectors in a lattice
- Solving the subset sum problem
- Finding roots of non-linear multivariate polynomial equations
- Solving the braid conjugacy problem

Intractable problems (contd)

Many sophisticated algorithms are proposed to break the trapdoor functions. Most of these are fully exponential. **Subexponential algorithms** are sometimes known.

For suitably chosen domain parameters these algorithms take infeasible time.

No non-trivial lower bounds on the complexity of these computational problems are known. Even existence of polynomial-time algorithms cannot be often ruled out. However, studies over several decades (or even centuries) failed to discover practical algorithms.

Certain special cases have been discovered to be cryptographically weak. For practical designs, it is essential to avoid these special cases.

Polynomial-time quantum algorithms are known for factoring integers and computing discrete logarithms in finite fields.

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

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

Multiplication: $ab \pmod{n} = (ab) \text{ rem } n$.

Inverse: $a \in \mathbb{Z}_n$ is called *invertible* modulo n if $(ua) \text{ rem } n = 1$ 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: $e = (e_{l-1}e_{l-2}\ldots e_1e_0)_2 = e_{l-1}2^{l-1}+e_{l-2}2^{l-2}+\cdots+e_12^1+e_02^0$.

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

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

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

Fermat's little theorem: Let $p \in \mathbb{P}$ and $a \in \mathbb{Z}$ with $p \nmid a$. 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 $\operatorname{ind}_3 15 = \operatorname{ind}_3 6 + \operatorname{ind}_3 11 = 15 + 7 = 6 \pmod{16}$.

The most common intractable problems of cryptography

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 only 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 $d = e^{-1} = 60343 \pmod{\phi(n)}$.

Public key: $(85067, 7)$.
Private key: $(85067, 60343)$.

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

Recover $m = c^d = (53384)^{60343} = 34152 \pmod{n}$.

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 $ed = 1 \pmod{\phi(n)}$.

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

Signature generation

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

Signature verification

Input: Signed message (m, s) and the signer's public key (n, e).
Output: "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 the size of n to be $\geqslant 1024$ bits 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.

ElGamal signature

Key generation

Like ElGamal encryption, one chooses p, g and computes a key-pair (y, d) where $y = g^d \pmod{p}$. The public key is (p, g, y), and the private key is (p, g, d).

Signature generation

Input: Message $m \in \mathbb{Z}_p$ to be signed and the signer's private key (p, g, d).
Output: The signed message (m, s, t).
Computation:

Generate a random session key $d' \in \{2, 3, \ldots, p-2\}$.
Compute $s = g^{d'} \pmod{p}$ and $t = d'^{-1}(H(m) - dH(s)) \pmod{p-1}$.

Signature verification

Input: A signed message (m, s, t) and the signer's public-key (p, g, y).
Computation:

Set $a_1 = g^{H(m)} \pmod{p}$ and $a_2 = y^{H(s)} s^t \pmod{p}$.
Output "signature verified" if and only if $a_1 = a_2$.

Correctness of ElGamal Signature

$H(m) = dH(s) + td'$. So $a_1 = g^{H(m)} = (g^d)^{H(s)}(g^{d'})^t = y^{H(s)}s^t = a_2 \pmod p$.

Example of ElGamal Signature

Take $p = 104729$ and $g = 89$. The signer chooses the private exponent $d = 72135$ and so $y = g^d = 98771 \pmod p$.

Let $m = 23456$ be the message to be signed. The signer chooses the session exponent $d' = 3951$ and computes $s = g^{d'} = 14413 \pmod p$ and $t = d'^{-1}(m - ds) = (3951)^{-1}(23456 - 72135 \times 14413) = 17515 \pmod{p-1}$.

Verification involves computation of $a_1 = g^m = 29201 \pmod p$ and $a_2 = y^s s^t = (98771)^{14413} \times (14413)^{17515} = 29201 \pmod p$. Since $a_1 = a_2$, the signature is verified.

A forger may choose any random d' (say, 3951 as above) and can compute $s = g^{d'} = 14413 \pmod p$. But computation of t involves d which is unknown to the forger. So the forger randomly selects $t = 81529$. Verification of this forged signature gives $a_1 = g^m = 29201 \pmod p$ as above. But $a_2 = y^s s^t = (98771)^{14413} \times (14413)^{81529} = 85885 \pmod p$, i.e., $a_1 \neq a_2$, and the forged signature is not verified.

Security of ElGamal signatures

Computation of s can be done by anybody. However, computation of t involves the signer's private exponent d. If the forger can solve the DLP modulo p, then d can be computed from the public-key y, and the correct signature can be generated.

The prime p should be large (of bit-size $\geqslant 1024$) in order to preclude this attack.

Some other encryption algorithms

Encryption algorithm	Security depends on
Rabin encryption	Square-root problem
Goldwasser-Micali encryption	Quadratic residuosity problem
Blum-Goldwasser encryption	Square-root problem
Chor-Rivest encryption	Subset sum problem
XTR	DLP
NTRU	Closest vector problem in lattices

Some other digital signature algorithms

Signature algorithm	Security depends on
Rabin signature	Square-root problem
Schnorr signature	DLP
Nyberg-Rueppel signature	DLP
Digital signature algorithm (DSA)	DLP
Elliptic curve version of DSA (ECDSA)	DLP in elliptic curves
XTR signature	DLP
NTRUSign	Closest vector problem in lattices

Blind signatures

A signer Bob signs a message m without knowing m.

Blind signatures insure anonymity in electronic payment schemes.

Chaum's blind RSA signature

Input: A message M generated by Alice.
Output: Bob's blind RSA signature on M.
Steps:

- Alice gets Bob's public-key (n, e).
- Alice computes $m = H(M) \in \mathbb{Z}_n$.
- Alice sends to Bob the masked message $m' = \rho^e m \pmod{n}$ for a random ρ.
- Bob sends the signature $\sigma = m'^d \pmod{n}$ back to Alice.
- Alice computes Bob's signature $s = \rho^{-1}\sigma \pmod{n}$ on M.

Correctness: Assume that $\rho \in \mathbb{Z}_n^*$. Since $ed = 1 \pmod{\phi(n)}$, we have $\sigma = m'^d = (\rho^e m)^d = \rho^{ed} m^d = \rho m^d \pmod{n}$. Therefore, $s = \rho^{-1}\sigma = m^d = H(M)^d \pmod{n}$.

Undeniable signatures

An active participation of the signer is necessary during signature verification.

A signer is not allowed to deny a legitimate signature made by him.

An undeniable signature comes with a **denial** or **disavowal protocol** that generates one of the following three outputs:

- Signature verified
- Signature forged
- The signer is trying to deny his signature by not properly participating in the protocol.

Examples

- Chaum-van Antwerpen undeniable signature scheme
- RSA-based undeniable signature scheme

Challenge-response authentication

Alice wants to prove to Bob her knowledge of the private key d in the key-pair (e, d).

Bob generates a random bit string r and computes $w = H(r)$.

Bob reads Alice's public key e and computes $c = f_e(r, e)$.

Bob sends the challenge (w, c) to Alice.

Alice computes $r' = f_d(c, d)$.

If $H(r') \neq w$, Alice quits the protocol.

Alice sends the response r' to Bob.

Bob accepts Alice's identity if and only if $r' = r$.

Correctness: Bob checks whether Alice can correctly decrypt the challenge c. Bob sends w as a **witness** of his knowledge of r. Before sending the decrypted plaintext r', Alice confirms that Bob actually knows the plaintext r.

The Guillou-Quisquater zero-knowledge protocol

Alice generates an RSA-based exponent-pair (e, d) under the modulus n.

Alice chooses a random $m \in \mathbb{Z}_n^*$ and computes $s = m^{-d} \pmod{n}$. Alice makes m public and keeps s secret. Alice tries to prove to Bob her knowledge of the secret s.

The protocol

Alice selects a random $c \in \mathbb{Z}_n^*$.	[Commitment]
Alice computes and sends to Bob $w = c^e \pmod{n}$.	[Witness]
Bob selects and sends to Alice a random $\epsilon \in \{1, 2, \ldots, e\}$.	[Challenge]
Alice computes and sends to Bob $r = cs^\epsilon \pmod{n}$.	[Response]
Bob computes $w' = m^\epsilon r^e \pmod{n}$.	
Bob accepts Alice's identity if and only if $w' \neq 0$ and $w' = w$.	

The Guillou-Quisquater zero-knowledge protocol (contd)

Correctness

$w' = m^{\epsilon} r^e = m^{\epsilon}(cs^{\epsilon})^e = m^{\epsilon}(cm^{-d\epsilon})^e = (m^{1-ed})^{\epsilon} c^e = c^e = w \pmod n$.

Security

The quantity s^{ϵ} is blinded by the random commitment c.

As a witness for c, Alice presents its encrypted version w.

Bob (or an eavesdropper) cannot decrypt w to compute c and subsequently s^{ϵ}.

An eavesdropper's guess about ϵ is successful with probability $1/e$.

The check $w' \neq 0$ precludes the case $c = 0$ which lets a claimant succeed always.

Digital certificates: Introduction

Bind public-keys to entities.

Required to establish the authenticity of public keys.

Guard against malicious public keys.

Promote confidence in using others' public keys.

Require a **Certification Authority** (CA) whom every entity over a network can believe. Typically, a government organization or a reputed company can be a CA.

In case a certificate is compromised, one requires to revoke it.

A revoked certificate cannot be used to establish the authenticity of a public key.

Digital certificates: Contents

A digital certificate contains particulars about the entity whose public key is to be embedded in the certificate. It contains:

- Name, address and other personal details of the entity.
- The public key of the entity. The key pair may be generated by either the entity or the CA. If the CA generates the key pair, then the private key is handed over to the entity by trusted couriers.

The certificate is digitally signed by the private key of the CA.

If signatures cannot be forged, nobody other than the CA can generate a valid certificate for an entity.

Digital certificates: Revocation

A certificate may become invalid due to several reasons:

- Expiry of the certificate
- Possible or suspected compromise of the entity's private key

An invalid certificate is revoked by the CA.

Certificate Revocation List (**CRL**): The CA maintains a list of revoked certificates.

If Alice wants to use Bob's public key, she obtains the certificate for Bob's public key. If the CA's signature is verified on this certificate and if the certificate is not found in the CRL, then Alice gains the desired confidence to use Bob's public key.

Part IV

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 take exponential (in $\log n$) running time.

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:

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

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 $\beta_i \in \{0, 1\}$, 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 $\gcd(x - y, n)$ yields 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 $\operatorname{ind}_g a$ 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 $d_i = \operatorname{ind}_g p_i$ 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:

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

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

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:

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

Side channel attacks

Applicable for both symmetric and asymmetric cryptosystems.

Relevant for smart-card based implementations.

Reveal secret information (key) by observing the decrypting/signing device.

Timing attack: utilizes reasonably accurate measurement of several private-key operations under the same key.

Power attack: analyzes power consumption patterns of the decrypting device during one or more private-key operations.

Fault attack: Random hardware faults during the private-key operation may reveal the key to an attacker. Even a single faulty computation may suffice.

Remedies: Shielding the decrypting device from external measurements, rechecking computations, adding random delays, etc.

Backdoor attacks

Suggested mostly for public-key cryptosystems.

The designer supplies a malicious key generation routine, so that published public keys reveal the private keys to the designer.

A good backdoor allows nobody other than the designer to steal keys.

Some backdoor attacks on RSA:

- Hiding prime factor
- Hiding small private exponent
- Hiding small public exponent

Backdoor attacks on ElGamal and Diffie-Hellman cryptosystems are also known.

Remedy: Use of trustworthy software (like open-source products).

Proving security of a cryptosystem

With our current knowledge, we **cannot** prove a practical system to be secure.

> *A standard security review, even by competent cryptographers, can only prove insecurity; it can never prove security. By following the pack you can leverage the cryptanalytic expertise of the worldwide community, not just a handful of hours of a consultant's time.*
>
> – Bruce Schneier, Crypto-gram, March 15, 1999

Desirable attributes for a *strong* cryptosystem:

- Use of good non-linearity (diffusion)
- Resilience against known attacks
- Computational equivalence with difficult mathematical problems

Cryptography: References

A. Das and C. E. Veni Madhavan, *Public-key Cryptography: Theory and Practice*, Pearson Education, in press.

A. J. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, Chapman & Hall/CRC, 1997.

W. Stallings, *Cryptography and Network Security: Principles and Practice*, third edition, Prentice Hall, 2003.

D. Stinson, *Cryptography: Theory and Practice*, third edition, Chapman & Hall/CRC, 2006.

J. Buchmann, *Introduction to Cryptography*, second edition, Springer, 2004.

B. Schneier, *Applied Cryptography*, second edition, John Wiley, 1996.

N. Koblitz, *A Course in Number Theory and Cryptography*, Springer, 1994.

H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*, Springer, 2002.

O. Goldreich, *Foundations of Cryptography*, two volumes, CUP, 2001, 2004.