Contents

1 Maths

1.1 Discrete Probability

Given a finite sample space $S = \{s_1, s_2, \ldots, s_n\}$ the probability P on S is $\{p_1, p_2, \ldots, p_n\}$ where:

$$0 \leq p_i \leq 1$$
$$\sum_i p_i = 1$$

1.1.1 Events

- Event E is a subset of S
- Complimentary event $\bar{E}$
- Probability of occurrence $P(E)$
- $P(E) = 1 - P(\bar{E})$

1.1.2 Joint Probability

- Probability of both events occurring
- $P(E_1 \cap E_2)$
- Events E_1 and E_2 are mutually exclusive if $P(E_1 \cap E_2) = 0$

1.1.3 Conditional Probability

Assuming $P(E_2) > 0$.
Probability of E_1 happening given that E_2 has happened:

$$P(E_1|E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)}$$

E_1 and E_2 are independent if $P(E_1|E_2) = P(E_1)P(E_2)$.

1.1.4 Bayes Theorem

Assuming $P(E_2) > 0$.

$$P(E_1|E_2) = \frac{P(E_1)P(E_2|E_1)}{P(E_2)}$$

1.2 Complexity

Polynomial time
Solved in $O(n^k)$

Exponential time
Cannot be solved in polynomial time

Decision problems that can be solved in polynomial time are tractable, otherwise they are intractable.

1.2.1 Complexity classes

P
Can be solved in polynomial time

NP
A positive answer can be verified in polynomial time

co-NP
A negative answer can be verified in polynomial time

NP-complete
Hardest problems in NP

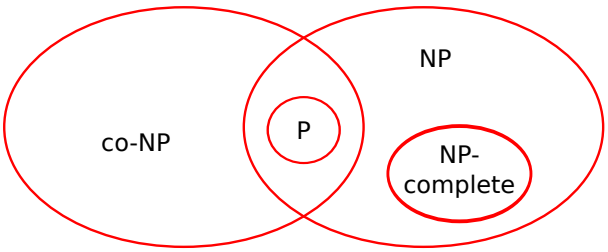


Figure 1: Complexity Classes

1.3 Number Theory

- $\mathbb{N}$ denotes positive integer such that $N \in \mathbb{N}^+$
- p denotes a prime number
- $\mathbb{Z}_N$ is the set of integers $\{0, 1, 2, \ldots, N-1\}$

1.3.1 Modular Arithmetic

Arithmetic module Z_m is the set $\{0, \ldots, m-1\}$ with operations $+$ and $\times$ such that:

1 Addition is closed:

$$\forall\, a, b \in Z_m, a + b \in Z_m$$

2 Addition is commutative:

$$\forall\, a, b \in Z_m, a + b = b + a$$

3 Addition is associative:

$$\forall\, a, b, c \in Z_m, (a + b) + c = a + (b + c)$$

4 0 is an additive identity:

$$\forall\, a \in Z_m, a + 0 = 0 + a = a$$

5 Additive inverse of any $a \in Z_m$ is $m - a$:

$$\forall\, a \in Z_m, a + (m - a) = (m - a) + a = 0$$

6 Multiplication is closed:

$$\forall\, a, b \in Z_m, ab \in Z_m$$

7 Multiplication is commutative:

$$\forall\, a, b \in Z_m, ab = ba$$

8 Multiplication is associative:

$$\forall\, a, b, c \in Z_m, (ab)c = a(bc)$$

9 1 is a multiplicative identity:

$$\forall\, a \in Z_m, a \times 1 = 1 \times a = a$$

10 The distributive property:

$$\forall\, a, b, c \in Z_m, (a + b)c = (ac) + (bc) \text{ and } a(b + c) = (ab) + (ac)$$

Items 1-5 establish Z_m is an abelian group, items 1-10 establish Z_m is a ring.

1.3.2 Greatest Common Denominator

$$\forall\, x, y \in \mathbb{Z}, \exists a, b \in \mathbb{Z} : a \cdot x + b \cdot y = gcd(x, y)$$

Integers x and y are coprime (relatively prime) if:

$$gcd(x, y) = 1 = a \cdot x + b \cdot y$$

Euclidean algorithm to compute GCD

- Inputs: integers x and y
- Output: $gcd(x, y)$
- $gcd(x, 1) = x$ as base case 1
- $\forall\, x, y \in \mathbb{Z}, gcd(x, y) = gcd(y, x \bmod y)$

1.3.3 Multiplicative Modular Inverse

Modular inverse of x is denoted by x^{-1}.
y is the multiplicative inverse of $x \bmod N$ if $x \cdot y = 1 (\bmod N)$.
x has a multiplicative inverse modulo N iff. $gcd(x, N) = 1$.
Compute using Euclidean algorithm:

$$\begin{aligned} a \cdot x + b \cdot N &= gcd(x, N) \\ a \cdot x &= 1 \in \mathbb{Z}_N \\ a &= x^{-1} \in \mathbb{Z}_N \end{aligned}$$

1.3.4 Invertible elements in $\mathbb{Z}_N$

The set of invertible elements is defined by:

$$(\mathbb{Z}_N)^* = \{x \in \mathbb{Z}_N : gcd(x, N) = 1\}$$

x^{-1} is the inverse of x if $x \cdot x^{-1} = 1$
If p is prime then $(\mathbb{Z}_p)^* = \mathbb{Z}_p/0$
$(\mathbb{Z}_p)^*$ is a cyclic group.

$$\exists g \in (\mathbb{Z}_n)^* : \{1, g, g^2, g^g, \ldots, g^{p-2}\} = (\mathbb{Z}_p)^*$$

g is generator of $(\mathbb{Z}_p)^*$, for example:

$$p = 5 : \{1, 3, 3^2, 3^3\} = \{1, 3, 4, 2\} = (\mathbb{Z}_5)^*$$

1.3.5 Solving linear equations

To solve:

$$a \cdot x + b = 0 \in \mathbb{Z}_N$$

1 Compute inverse a^{-1}

2 Subtract b

3 Multiply by inverse a^{-1}

Solution:

$$x = -b \cdot a^{-1} \in \mathbb{Z}_N$$

1.3.6 Fermat's Little Theorem

$$\forall\, x \in (\mathbb{Z}_p)^* :\ x^{p-1} = 1 (mod\, p)$$

Calculating inverses:

$$x^{-1} = x^{p-2} (mod\, p)$$

Generating large prime numbers:

1 Choose random base a such that $2 \leq a < p - 1$

2 Choose a random candidate number p of required bit length

3 Test $a^{p-1} = 1 (mod\, p)$, if test passes the p is prime

1.3.7 Euler's ϕ function

The number of invertible elements in $\mathbb{Z}_n$.

$$\phi(n) = |(\mathbb{Z}_\ltimes)^*|$$

1.3.8 Group order

Order of g in $(\mathbb{Z}_p)^*$ is the size of the group $< g >$.

Raising a generator g to its order c gives 1:

$$g^c = 1 (mod\, p)$$

1.3.9 Euler's Theorem

$$\forall\, x \in (\mathbb{Z}_N)^* :\ x^{\phi(N)} = 1\ (mod\, N)$$

Can be used to cancel out elements where the group order in in an exponent, useful to remove the encryption operation.

1.3.10 eth root

eth root of c ($c^{1/e}$) is $x \in (\mathbb{Z}_p)^*$:

$$x^e = c\ (mod\, p)$$

Computing eth root

Easy if two conditions are met:

1 Operating in $(\mathbb{Z}_p)^*$

2 e is coprime to $p - 1$

If both conditions are true eth root can be computed by:

1 Compute the inverse d of e

$$d = e^{-1}\ (mod\, p - 1)$$

2 Compute the eth root using c^d

1.3.11 Discrete Logarithm

Given $h \in (\mathbb{Z}_p)^*$ and generator g, find x such that $g^x = h$.

1.4 Intractability

Intractability of a problem is tested used a setup defining the inputs and outputs to/from an adversary and the criteria in which the adversary is successful.

A problem is hard when probabilistic and polynomial time adversaries have only a small chance of success.

1.4.1 Factoring

Problem Find p and q given $N = p \cdot q$

Over $\mathbb{Z}$

Generation $p \cdot q$

Solution Factorise N

Used For RSA encryption, RSA signatures

Proof:

Setup
(N, p, q) such that $p \cdot q = N$

Inputs
N

Outputs
p' and q'

Success criteria
$p' \cdot q' = N$

1.4.2 eth root (RSA)

Problem Find x given $x^e = y \,(mod\, N)$

Over $(\mathbb{Z}_n)^*$

Generation $x^e \,(mod\, N)$

Solution Find eth root x given y

Used For RSA encryption, RSA signatures

Proof:

Setup
(N, e, d) such that $e \cdot d = 1 \,(mod\, \phi(N))$
Choose y from $(\mathbb{Z}_N)^*$

Inputs
N, e, y

Outputs
$x \in (\mathbb{Z}_N)^*$

Success criteria
$e^x = y \,(mod\, N)$

1.4.3 Discrete Logarithms

Problem Find x given $g^x = h \,(mod\, p)$

Over Subgroups of $(\mathbb{Z}_p)^*$

Generation $g^x \,(mod\, p)$

Solution Find discrete log x given h

Used For ElGamal encryption, DSA signatures, Schnorr

Proof:

Setup
$((\mathbb{Z}_p)^*, q, g)$ such that $q = order(g)$
Choose h from $(\mathbb{Z}_p)^*$ such that $h = g^x \ (mod\, p)$

Inputs
$(\mathbb{Z}_p)^*$, q, g, h

Outputs
$x \in (\mathbb{Z}_q)^*$

Success criteria
$g^x = h \ (mod\, p)$

1.4.4 Diffie-Hellman decision

Problem Distinguish g^{xy} from g^z

Over Subgroups of $(\mathbb{Z}_p)^*$

Generation $g^{xy} \ (mod\, p)$

Solution Decide if K is DH or random

Used For ElGamal encryption, DSA signatures, Schnorr

Proof:

Setup
$((\mathbb{Z}_p)^*, q, g)$ such that $q = order(g)$
Choose $h_1 = g^x \ (mod\, p)$ and $h_2 = g^y \ (mod\, p)$
Choose at random $K = g^z$ or $K = g^{xy}$

Inputs
$(\mathbb{Z}_p)^*$, q, g, h_1, h_2, K

Outputs
Decision if $K = g^z$ or $K = g^{xy}$

Success criteria
If decision about K is correct

2 Symmetric

2.1 Information Theory

2.1.1 Entropy

Let X be a random variable $\in \{x_1, x_2, \ldots, x_n\}$ with probabilities $P = \{p_1, p_2, \ldots, p_n\}$.
The entropy (uncertainty) of X is defined as:

$$H(X) = -\sum_i p_i log_2(p_i)$$

2.1.2 Rate of a language

Given language M of length N, the rate of M is:

$$r = H(M)/N$$

2.1.3 Redundancy of English language

- $N = 26$
- Assume even probability distribution
- Each latter represents $log_2(26) = 4.3$ bits
- Taking into account actual distribution each letter only contains 1.3 bits

2.1.4 Confusion and Diffusion

Two techniques for obscuring a plain text message.

Confusion
Obscures relationship between plain text and cipher text.
e.g. Character substitution

Diffusion
Spreads plain text message throughout cipher text to remove patterns.
i.e. Changing a single bit of the plain text changes multiple bits of the cipher text

2.1.5 Perfect Secrecy

The cipher text yields no possible information about the plain text (except from the length).

$$P(m|c) = P(m) \, \forall \, m \in M, c \in C$$

Requires number of keys $\geq$ number of plain text messages.
Only possible with the one time pad (see 2.3.1).

2.2 Classical Ciphers

2.2.1 Shift cipher

Every letter is rotated K positions in the alphabet.

Encryption:

$$e_K(x) = (x + K) \; mod \, 26$$

Decryption:

$$d_K(x) = (x - K) \; mod \, 26$$

For $0 \leq K \leq 26$ and $x, y \in \mathbb{Z}_{26}$.

Caeser cipher $K = 3$

ROT13 $K = 13$

Vulnerabilities

- Can easily be broken by exhaustive search
- Only 26 keys (K) for English alphabet

2.2.2 Substitution cipher

Replace each letter with a given substitution defined by permutation π.

K contains all permutations π of the English alphabet.

Encryption:

$$e_\pi(x) = \pi(x) \; mod \, 26$$

Decryption:

$$d_\pi(x) = \pi^{-1}(x) \; mod \, 26$$

For $0 \leq K \leq 26$ and $x, y \in \mathbb{Z}_{26}$.

Key space $|K| = 26!$.

Vulnerabilities

- Can easily be broken by frequency analysis
- Probability distribution of English characters is well defined
- This distribution can be remapped to derive the permutation
- Only provides good confusion (and not diffusion)

2.2.3 Vigenère cipher

Combining several shift ciphers.

k	A	B	C	A	B	C	A	B	C	A	B	C
m	B	E	R	E	A	D	Y	A	K	K	A	T
c	C	G	U	F	C	G	Z	C	F	L	C	W

Table 1: Vigenère cipher example

Cryptanalysis involves:

1 Finding key length m (number of shift ciphers)

2 Breaking each individual shift cipher to obtain each character in the key

Obtaining key length: Kasiski test

Search for identical segments in cipher text and count how far apart they are.

Obtaining key length: Index or coincidence

Probability that two elements of of a string of characters are identical.

$$I_c(x) = \sum_{i=1}^{26} p_i^2$$

where p_i is the probability of character i in text string x.

Can then compare I_c with known values for different key lengths.

Breaking shift ciphers

Break each individual shift cipher using frequency analysis.

Vulnerabilities

- Transposition does not randomly distribute information in cipher text
- A secure cipher text should not contain any decipherable pattern

2.3 Stream Cipher

Encrypt individual characters one at a time.

2.3.1 One time pad

$$c_i = k_i \oplus m_i$$

- Perfect secrecy
- Requires key size $\geq$ message size
- Key can only be used for a single encryption
- Perfect but impractical

Idea of stream ciphers: make one time pad practical by generating a long key stream using a short secret key.

2.3.2 Synchronous stream cipher

Construct a key stream form a short key.

Assume a key of length m bits. A key stream can be generated with a linear recurrence of degree m.

e.g.:

$$k_{i+m} = \sum_{j=0}^{m-1} c_j k_{i+j} \, mod \, 2$$

where $c_0, \ldots, c_{m-1}$ are constants.

After a period the key stream will repeat.

This key stream generation can be implemented in application specific hardware to make stream generation much faster, such an implementation is known as a Linear Feedback Shift Register.

In a synchronous stream cipher the insertion or deletion of cipher text will cause decryption to fail (the cipher text and key streams will be out of sync).

2.3.3 Vulnerabilities/Attacks

Two time pad

Possible when two different plain text messages are encrypted with the same one time pad.

Can obtain the XOR of plain text messages:

$$\begin{aligned} c_1 &= m_1 \oplus f(k) \\ c_2 &= m_2 \oplus f(k) \\ c_1 \oplus c_2 &= m_1 \oplus m_2 \\ &\rightarrow \{m_1, m_2\} \end{aligned}$$

No integrity

Problem for all stream ciphers.

Can modify cipher text without detection by the recipient, such that the plain text is changed in some meaningful way.

Weakness in algorithm

Poorly developed (proprietary) algorithms may have vulnerabilities not found in more common, widely used algorithms.

e.g. Content Scramble System (CSS) used to encrypt contents of DVDs.

40 bit key using two LFSRs (17 bit and 25 bit), disk key could be retrieved in 2^{25} time. Only provided 17 bits of security.

2.4 Block Cipher

Encryption of a fixed size block of data.

2.4.1 Data Encryption Standard (DES)

- 64 bit block size
- Implemented using 16 round Feistel network

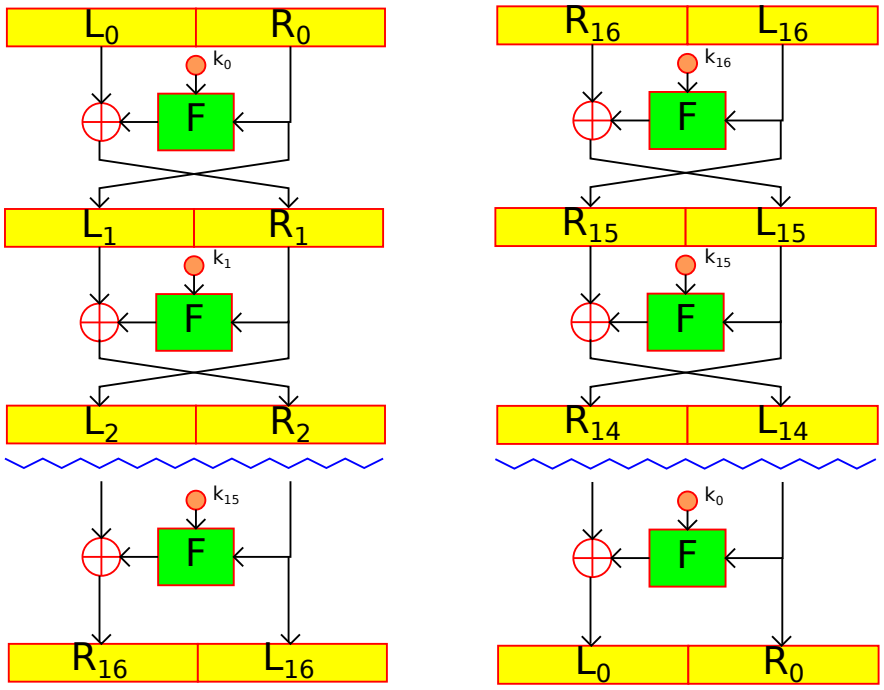
- Use the same circuit to decrypt (possible with the last swap being removed)

Encryption:

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(K_i, R_i)$$

Decryption:

$$R_i = L_{i+1}$$
$$L_i = R_{i+1} \oplus F(K_i, R_i)$$
$$= R_{i+1} \oplus F(K_i, L_{i+1})$$



(a) Encryption (b) Decryption

Figure 2: Feistel Networks

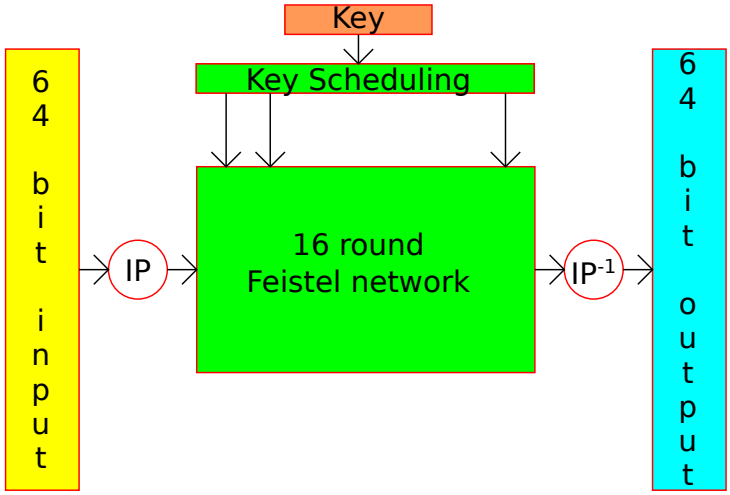


Figure 3: DES overview

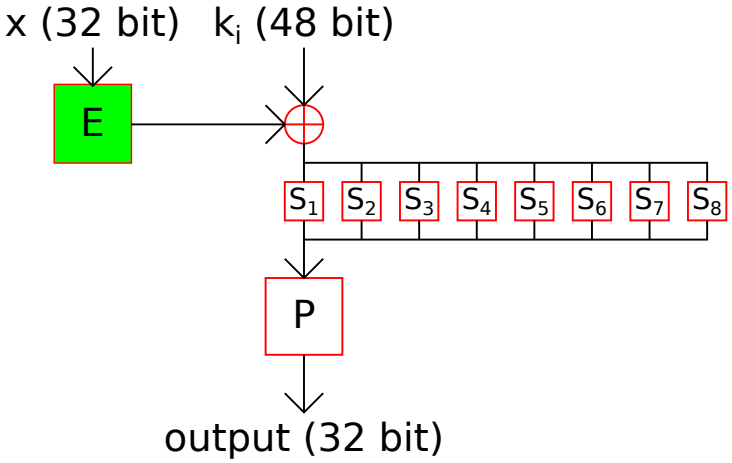


Figure 4: F function

- F function adds confusion and diffusion
- S box (look up table) adds confusion
- P box (look up table) adds diffusion

2.4.2 Triple DES

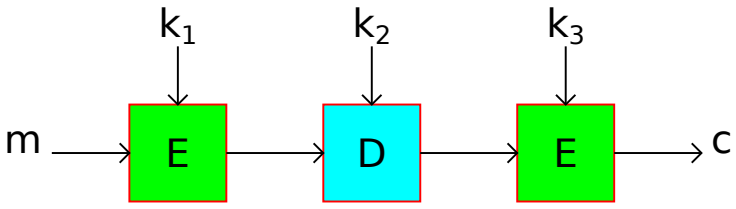


Figure 5: Triple DES

- Does not triple security
- Design is compatible with standard DES

2.4.3 Modes of Operation

Defines how a block cipher E is applied to encrypt data.

Electronic Code Book (ECB)

- Simplest mode of operation
- Message split into several blocks and each block encrypted individually
- Deterministic
- Possibility of leaking data through patterns in cipher data

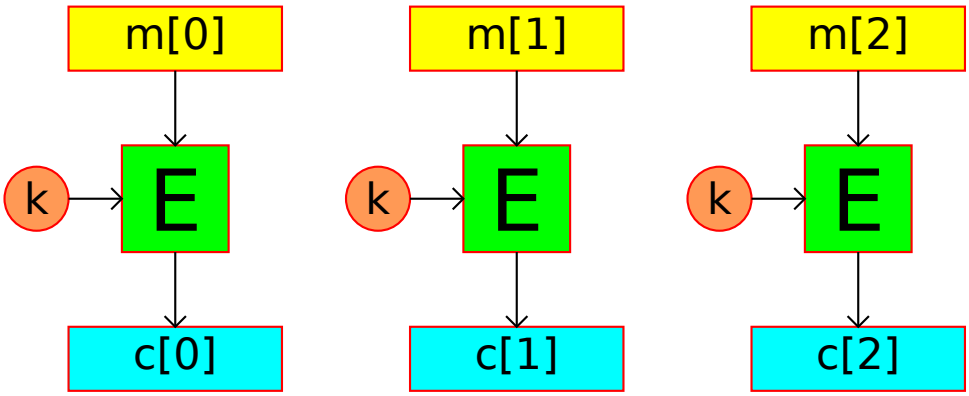


Figure 6: Electronic Codebook

Cipher Block Chaining (CBC)

- Previous cipher text feeds back to next block
- Padding added to ensure correct block size
- Random initialisation vector IV used to ensure non-deterministic output
- Can resynchronise if a block is lost (not if a single byte is lost)
- Cannot be parallelised
- Widely used

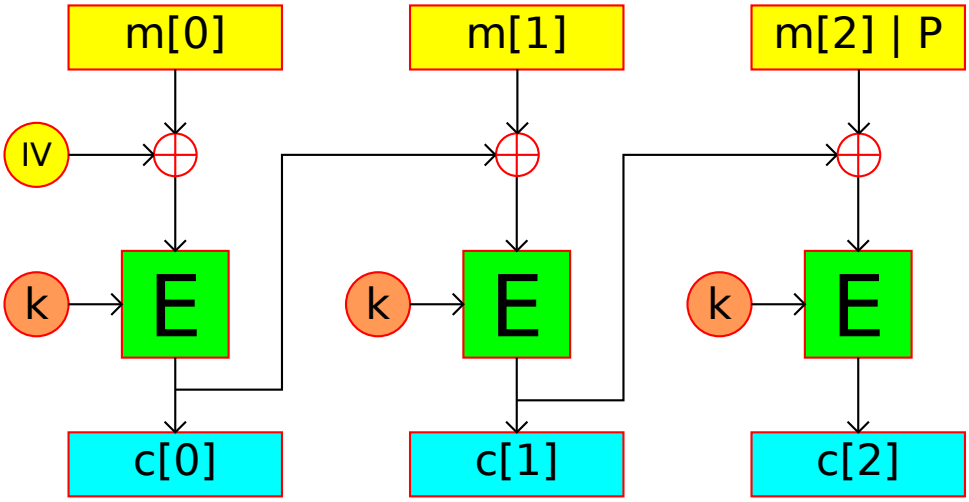


Figure 7: Cipher Block Chaining

Cipher Feedback (CFB)

- Essentially a stream cipher
- Can resynchronise if a block is lost (not if a single byte is lost)
- Encryption cannot be parallelised
- Decryption is a single E operation per block (can be parallelised)

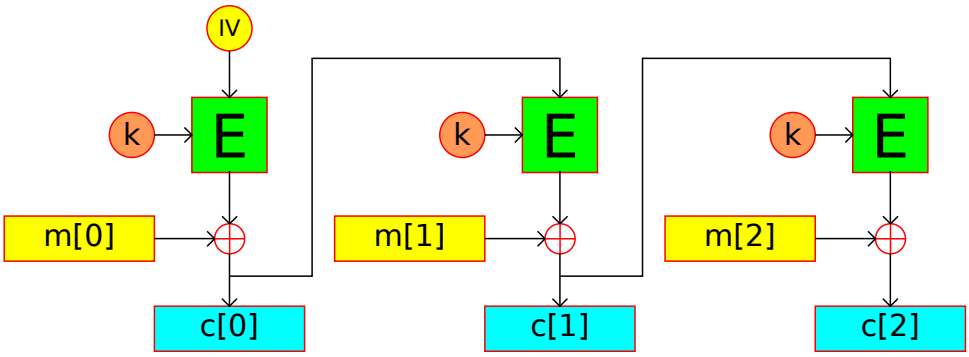


Figure 8: Cipher Feedback

Output Feedback (OFB)

- Essentially a stream cipher
- Identical encryption and decryption operations

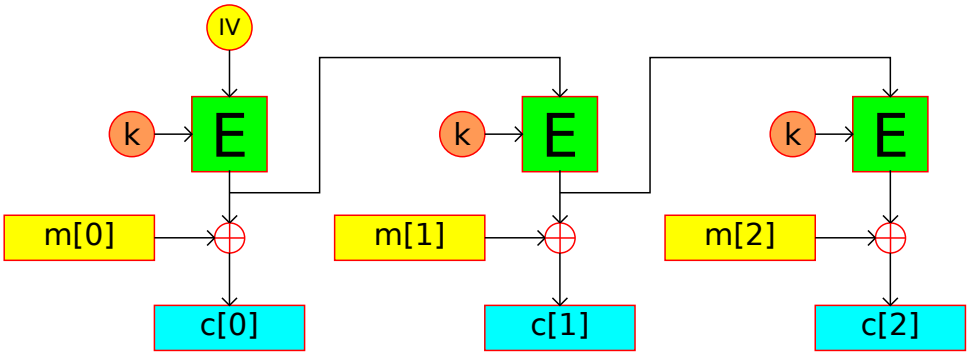


Figure 9: Output Feedback

Counter (CRT)

- Essentially a stream cipher
- Uses an incrementing counter i instead of initialisation vector
- Identical encryption and decryption operations
- Both encryption and decryption can be parallelised
- Popular as a replacement for CBC

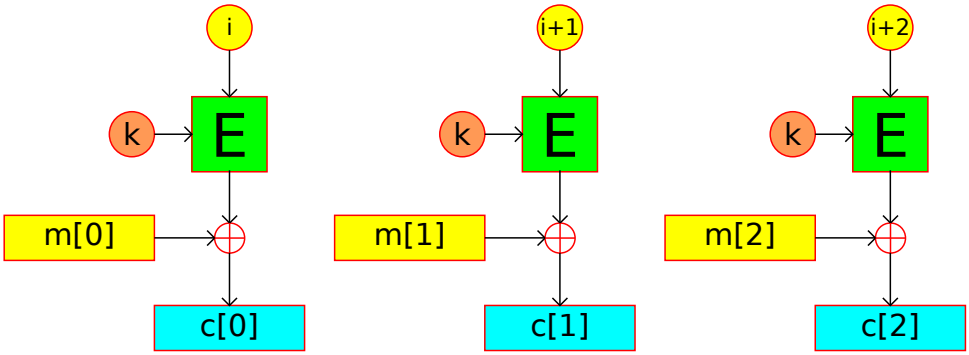


Figure 10: Counter

2.5 Hash Function

- Compress an arbitrary message into a fixed length output.
- Historically used for detecting data defects or equality
- Cryptographic hash used to verify integrity

2.5.1 Random Oracle Model

- Ideal hash function
- Function generates a unique random hash for every new request
- Maintains database of existing hashes for duplicate requests
- No hash collisions
- Not practical to implement

2.5.2 Security Requirements

1 Pre-image resistance
Given $H(m)$, cannot obtain m

2 Second pre-image resistance
Given $H(m_1)$ cannot find m_2 such that $H(m_1) = H(m_2)$

3 Collision resistance
Cannot obtain unique m_1 and m_2 such that $H(m_1) = H(m_2)$

2.5.3 Birthday attack (collision resistance)

Assume a hash function with n bits output.

1 Select $2^{n/2}$ random input messages

2 Compute hash of each message

3 Look for a collision, if not found repeat until a collision occurs

For n bit security, hash output must be at least $2n$ bits long.

2.5.4 Design considerations

- Operational mode
- Compression function
- Confusion-diffusion operations

2.5.5 Construction: Merkle-Damgard

- Start with a fixed initialisation vector IV
- Cascade several compression functions, f
- f operates over fixed length subsections of the message m
- Padding P is added to ensure correct length

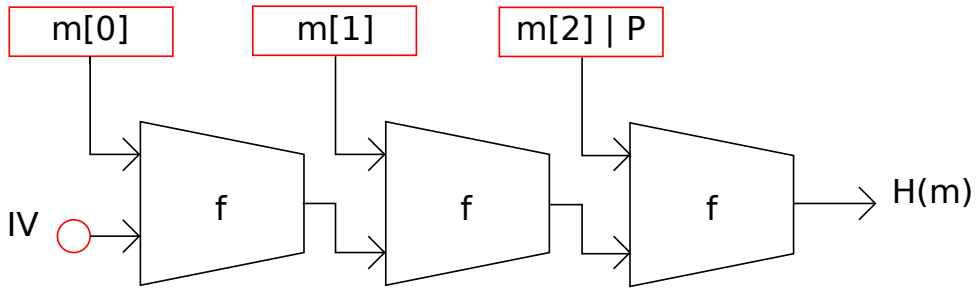


Figure 11: Merkle-Damgard construction

Theorem is that if f is collision resistant then H is also collision resistant.

2.5.6 Compression function: Davies-Meyer

E is a block cipher using output of previous f (or IV if is first block) as message and $m[i]$ as key.

Defined as:

$$f(H, m) = E(m, H) \oplus H$$

Compression:

$$|input| = |key| + |block|$$
$$|output| = |block|$$

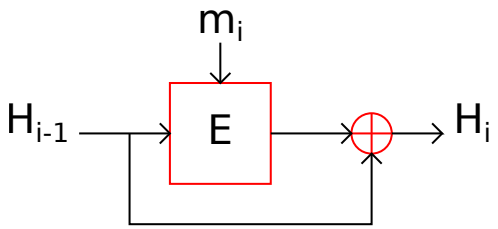


Figure 12: Davies-Meyer compression function

2.5.7 Applications

- Digital signatures
- Integrity checking
- Random number generator, random function
- Password masking/storage

2.5.8 Salt

Hashing passwords with a salt is better practice, this makes it more difficult for an attacker to recover the plain text password.

Dictionary attack

Given $H(password, salt)$ and $salt$ an attacker can do an exhaustive search for all passwords and obtain the correct one.

Feasible since passwords have low entropy.

2.6 MAC

- Used to verify integrity of a message
- Sender generates a tag t using function S and secret key k
- Recipient generates tag from message and k using S and compares to the tag sent by the sender
- Use of a secret key ensures the tab cannot simply be recomputed by an attacker

2.6.1 Security

Attacker can use a chosen message attack to try to obtain the key k.

Given a set of messages $M = \{m_1, m_2, \ldots, m_n\}$ and their tags $T = \{t_1, t_2, \ldots, t_n\}$ such that $t_i \leftarrow S(k, m_i)$.

Attacker's goal is existential forgery, i.e. producing a new message/tag pair $(m, t) \in \{M, T\}$. In this case the attacker can successfully forge a tag that the recipient would believe is valid.

2.6.2 Construction 1: block cipher based

Cipher Block Chaining (CBC) method.

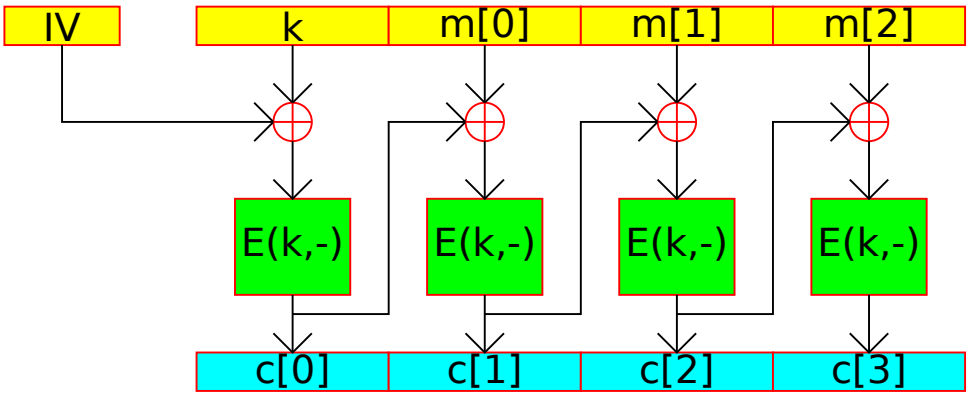


Figure 13: CBC-MAC

MAC is given by $c[3]$.

2.6.3 Construction 2: hash function based

Cannot simply use the hash function $H(k, m)$ as an additional message block can be added to the existing hash (see figure 11).

HMAC

Needs to have different secret keys to protect both the front and back of the hash.

e.g. $H(k_1, H(k_2, m))$

In HMAC only one key is used for efficiency and two constants O_{pad} and I_{pad} are used to derive k_1 and k_2.

$$HMAC(k, m) = H(k \oplus O_{pad} || H(k \oplus I_{pad} || m))$$

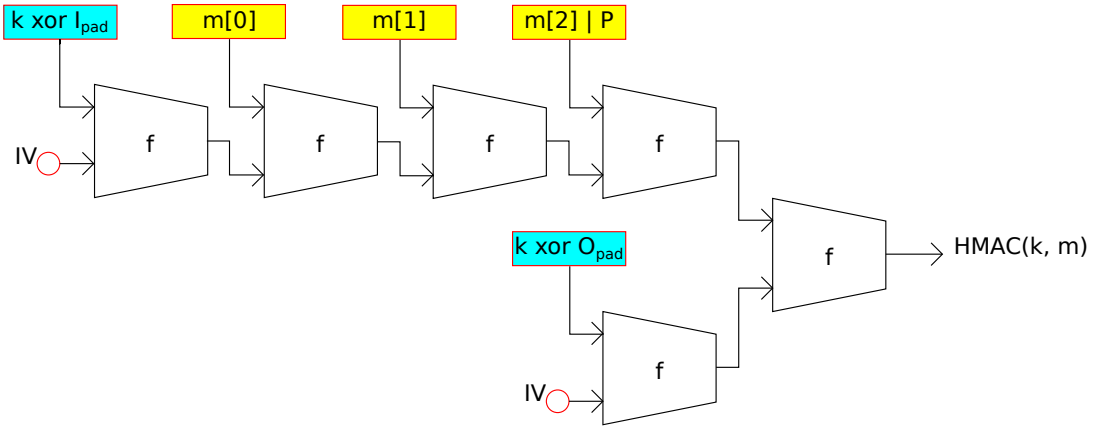


Figure 14: HMAC

2.6.4 Verification timing attack

Some cryptographic libraries implement MAC verification with a byte by byte comparison which returns false at the first incorrect byte.

By measuring the time it takes MAC verification to fail an attacker can determine which byte of the MAC the verification failed on, therefore allowing them to obtain the MAC by exhaustive search.

The solution is to make the verification always take the same time, the simplest way is to perform comparison of all bytes regardless of any previous inequalities.

2.6.5 Authenticated encryption

In real world applications encryption often takes place in authenticated mode, as part of this a MAC is generated during the encryption process.

This provides both confidentiality (through the encryption) and integrity (through the MAC).

3 Asymmetric

3.1 Key Exchange

- Need to transmit a secret key on an open channel without eavesdroppers learning the key
- Too many keys to manage for each pair of participants to have their own key ($\mathcal{O}(n^2)$)
- Ad-hoc method of key exchange is required in order to establish a secure communication channel between two parties

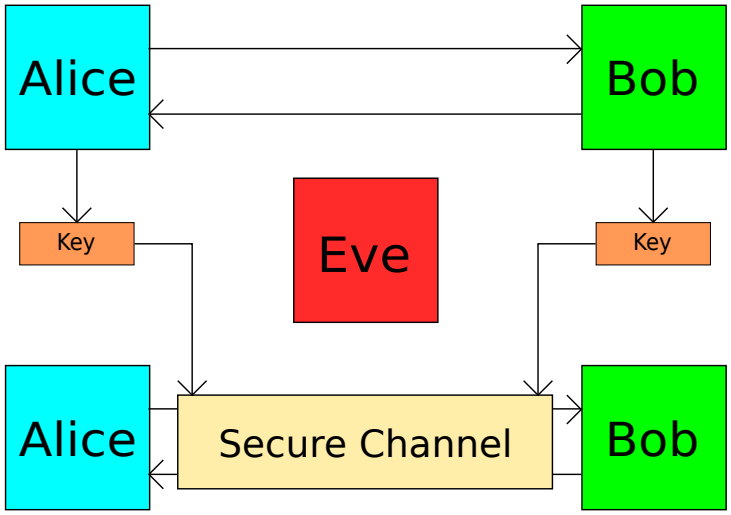


Figure 15: Key Exchange Problem

3.1.1 Trusted Third Party

- Only $\mathcal{O}(n)$ keys required
- No third party can be truly/fully trusted

3.1.2 Merkle Puzzles

- Only secure against eavesdropping
- Only uses symmetric primitives
- Not used in real world applications

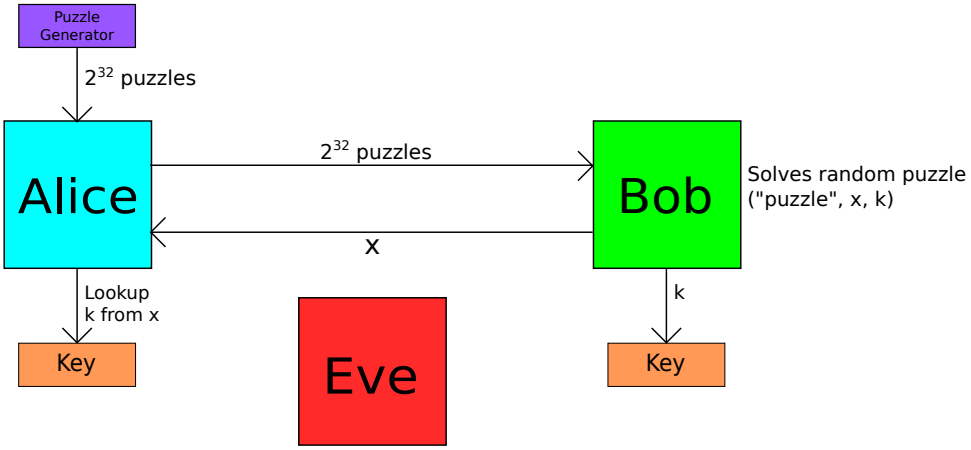


Figure 16: Merkle Puzzles Key Exchange

Idea

1 Alice generates a large number of puzzles (e.g. 2^{32}) each with a candidate session key in the payload

2 Alice sends all the puzzles to Bob

3 Bob chooses a puzzle at random and solves it by brute force

4 Bob sends the puzzle ID back to Alice, Alice then knows which puzzle hence which session key Bob is using

5 Alice sends a random number N to Bob, encrypted with the chosen session key

6 Bob replies with $N-1$ encrypted using the chosen session key

7 If the message from bob is actually $N-1$ then the key exchange was successful

Choose AES key with leading zeros such that:

$$K = \{0\}^{128-l}\{0,1\}^{l}$$

Requires 2^l brute force decryption attempts to solve.

Puzzle generator

1 Choose random $P \in \{0,1\}^{32}$

2 Choose random $x, k \in \{0,1\}^{128}$

3 Puzzle $P = E((\{0\}^{96}||P), "puzzle"x, k)$

Puzzle structure

$\{0\}^{96}||P$
: Random encryption key

$"puzzle"x$
: Random ID

k
: Random session key

Computational gap

- Workload for Alice and Bob: $\mathcal{O}(n)$, e.g. 2^{32}
- Workload for Eve: $\mathcal{O}(n^2)$, e.g. 2^{64}
- Quadratic gap is best achievable for symmetric block ciphers

3.1.3 Diffie-Hellman

- Only secure against eavesdropping
- Uses asymmetric primitives

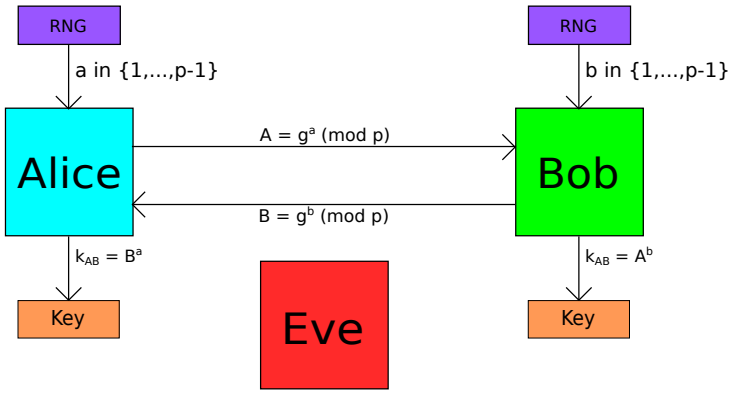


Figure 17: Diffe-Hellman Key Exchange

Idea

1 A large prime number p and integer g (generator of $(\mathbb{Z}_p)^*$) are known by everyone

2 Alice chooses a random $a \in \mathbb{Z}_p/\{0\}$

2 Bob chooses a random $b \in \mathbb{Z}_p/\{0\}$

3 Alice transmits $A = g^a (\,mod\, p)$ to Bob

4 Bob transmits $B = g^b (\,mod\, p)$ to Alice

5 Alice computes session key $k_{AB} = B^a$

6 Bob computes session key $k_{AB} = A^b$

Proof of correctness

Proof of correctness of session keys:

$$\begin{aligned} k_{AB(alice)} &= k_{AB(bob)} \\ B^a &= A^b \\ g^{b^a} &= g^{a^b} \\ g^{ba} &= g^{ab} \\ g^{ab} &= g^{ab} \end{aligned}$$

Difficulty

Symmetric Cipher Key Size	DH/RSA Modulus Size
80 bits	1028 bits
128 bits	3072 bits

Table 2: Diffie-Hellman Difficulty

Best known algorithm to break DH/RSA is General Number Field Sieve, the expected running time of which is $\mathcal{O}(exp(log(N)^{\frac{1}{3}}))$.

3.1.4 Establishing a secure channel

- Messages are sent encrypted with a symmetric cipher with a MAC
- Avoid using same key for all ciphers and MAC generation by using key derivation

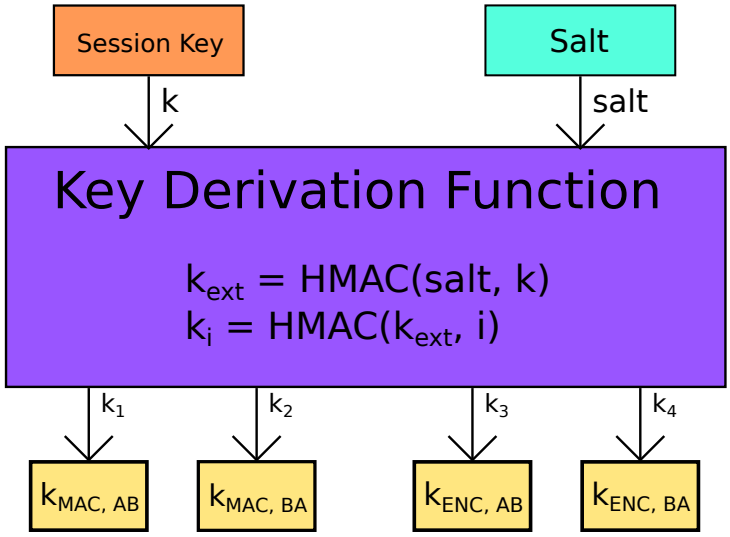


Figure 18: Key Derivation Function

Key derivation generates different keys for cipher and MAC in both directions (4 unique keys in total) using the session key and a known salt.

3.2 Public Key Encryption

- Encrypt using public key
- Decrypt using secret key
- Only one secret key to keeps secure
- Slower then symmetric encryption

Key points:

- Key distribution is authentic
- Cipher texts are indistinguishable from random
- Randomized
- Efficiency with hybrid encryption

3.2.1 Chosen Plaintext Attack Security

Proof of chosen plaintext attack security:

1 Eve chooses two messages m_o and m_1

2 Alice chooses one of two messages to encrypt and sends ciphertext c to Eve (selected by generating a random bit b)

3 Eve determines which message the c was generated from

4 Eve sends b' to Alice denoting which message Eve believes c was generated form

5 Eve is successful if $b = b'$

3.2.2 Hybrid Encryption

Use an asymmetric cipher to distribute a symmetric cipher key between two parties, then use symmetric encryption to transfer large amounts of data.

Combines efficiency of asymmetric encryption with convenience of asymmetric.

3.2.3 RSA

Based on:

- Factoring
- eth root

Key Generation: GenRSA(1^n)

- Input:
  Key length n
- Generate two large n bit primes p and q
- Compute $N = p \cdot q$
- Compute $\phi(N) = (p-1) \cdot (q-1)$
- Choose a random integer e : $gcd(e, \phi(N)) = 1$
- Compute inverse of e, $e \cdot d = 1 \, (mod \, \phi(N))$
- Output:
  Secret key (N, e)
  Public key (N, d)

Encryption:

- Input:
  Public key (N, e)
  Message m
- $c = m^e$

Decryption:

- Input:
  Secret key (N, e)
  Cipher text c
- $m = c^d$

Proof:

$$\begin{aligned} Dec_{sk}(Enc_{pk}(m)) &= m \\ c &= m^e \\ c^d &= (m^e)^d \\ &= m^{de} \\ &= m^{1+k\phi(N)} \\ &= m^1 \cdot m^{k\phi(N)} \\ &= m \cdot 1 = m \end{aligned}$$

It is theoretically possible to obtain d given public key (N, e), however the method would involve factoring N (a difficult problem) ad no such method currently exists.

Textbook RSA is not secure, attacks include:

Encryption with small e
If both e and m are small then the encryption does not "wrap around" the modulus N.
Can then compute eth root over integer to decrypt message.

Common modulus attack
If the same modulus N is used throughout an organisation then it is possible for one user to obtain the private keys for all other users.

$$\begin{aligned} e \cdot d &= 1 \,(mod\, \phi(N)) \\ &= 1 + k\phi(N) \end{aligned}$$

Can then compute d for any user using the inverse of their public exponent e using:

$$e \cdot d = 1 \,(mod\, \phi(N))$$

Ciphertext mangling
No integrity built in so a cipher text can be modified without detection.

Padding is used to ensure integrity (two methods listed: PKCS & OAEP).

Messages are encoded with a structure and random padding before encryption.

When decrypting check that the structure is intact, changes to the structure indicate that the cipher text has been tampered with.

PKCS

Byte structure:

$$(00000000|00000010|r|00000000|m)^e \,(mod\, N)$$

Structure is:

- 16 bit integer 2
- Random padding r
- 8 bit integer 0
- Message m

OAEP

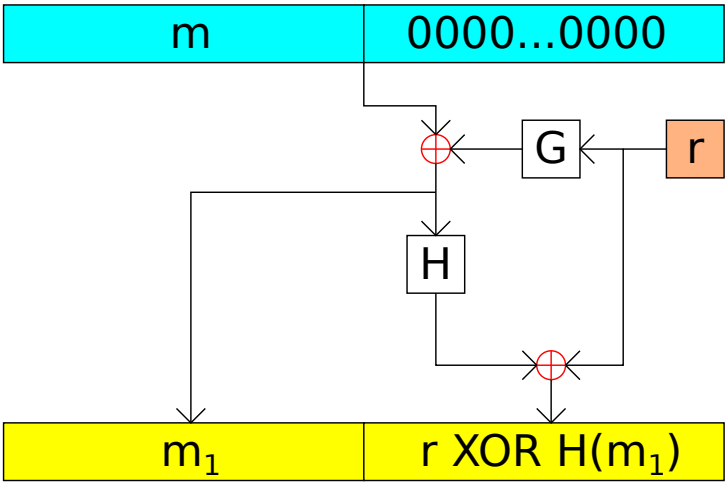


Figure 19: RSA OAEP encryption

- Operates in module N
- Uses two hash functions G and H
- Message can only be half modulo size, remainder of modulo is padded with zeros

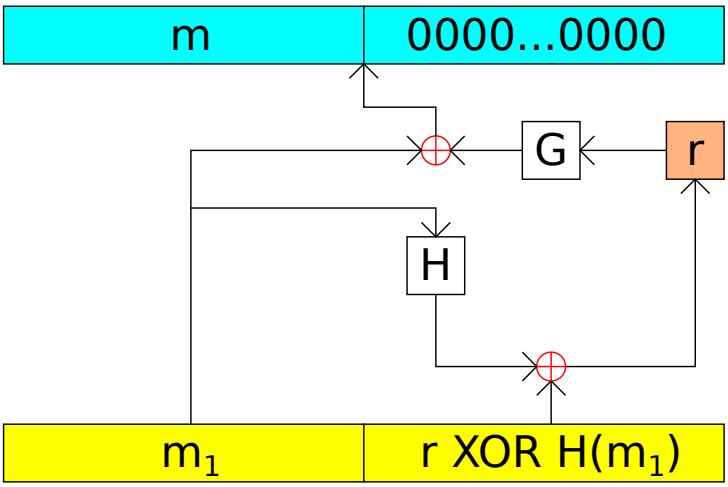


Figure 20: RSA OAEP decryption

3.2.4 ElGamal

Based on:

- Discrete logarithm problem
- Diffie-Hellman problem

Key generation: $GenElGamal(1^n)$

- Input:
  Key length n

- Generate a cyclic group G, with order q and generator g
- Choose random $x \in \mathbb{Z}_p$
- Compute $h = g^x \ (mod\, q)$
- Output:
  Public key: (G, q, g, h)
  Secret key: (G, q, g, x)

Encryption:

Choose random $y \in \mathbb{Z}_q$.

$$c_1 = g^y$$
$$c_2 = h^y \cdot m$$

Decryption:

$$m = c_2 \cdot (c_1^x)^{-1}$$

Proof:

$$\begin{aligned} m &= c_2 \cdot (c_1^x)^{-1} \\ &= m \cdot h^y \cdot (c_1^x)^{-1} && (c_2 = h^y \cdot m) \\ &= m \cdot h^y \cdot (g^{xy})^{-1} && (c_1 = g^y) \\ &= m \cdot g^{xy} \cdot (g^{xy})^{-1} && (h = g^x) \\ &= m \cdot 1 = m && (a \cdot a^{-1} = 1) \end{aligned}$$

Attack on weak randomness

If the same y is used to encrypt multiple messages m_a and m_b then the cipher texts can be divided to obtain:

$$\frac{c_{2a}}{c_{2b}} = \frac{h^y \cdot m_a}{h^y \cdot m_b} = \frac{m_a}{m_b}$$

3.3 Digital Signatures

- Goal: integrity
- Public verifiability
  Anyone with access to public key can verify a signature
- Transferability
  One can convince others that the key is valid
- Non-repudiation
  Alice can not dispute that she has signed the message
- Key authenticity
  Public key is distributed with integrity

A MAC is not equivalent of a digital signature:

- Signatures have no pre shared secret (do not need key exchange)
- Signatures can be verified by anyone
- A signature only requires one secret key
- Signatures have non-repudiation
- (Signatures are slower then MAC)

3.3.1 Existential Unforgeability

Setup
Key pair: (pk, sk)

Inputs
pk, signing function $S_{sk}()$ and set of messages $m \in Q$

Outputs
Message-signature pair: (m', σ)

Success criteria
$m' \notin Q$

3.3.2 RSA

Key generation: $GenRSA(1^n)$ (see 3.2.3)

Sign:

$$\sigma = m^d \, (mod \, N)$$

Verify:

$$m = \sigma^e \, (mod \, N)$$

No message attack

Adversary only has access to a public key $pk = (N, e)$.

Use RSA as encryption scheme:

1 Select random signature $\sigma \in (\mathbb{Z}_N)^*$

2 Compute message $m = \sigma^e$

Selected message attack

Adversary has access to public key $pk = (N, e)$ and can obtain two signatures on messages of their choosing.

To forge a signature on message m:

1 Choose random message $m_1 \in (\mathbb{Z}_N)^*$

2 Compute $m_2 = m/m_1 \, (mod \, N)$

3 Obtain signatures σ_1, σ_2 for messages m_1, m_2

4 Compute signature σ for m: $\sigma = \sigma_1 \cdot \sigma_2 \, (mod \, N)$

(same principle as mangling ciphertext attack used in RSA encryption)

3.3.3 Digital Signature Standard (DSS)

Key generation: $GenDSS(1^n)$

- Input:
  Key length n
- Generate a cyclic group G, with order q and generator g

- Choose random $x \in \mathbb{Z}_p$
- Compute $y = g^x \ (mod\, q)$
- Output:
  Public key: (G, q, g, y)
  Secret key: (G, q, g, x)

Sign:

- Choose random $k \in \mathbb{Z}_q$

$$r = g^k \ (mod\, p) \ (mod\, q))$$
$$s = (H(m) + xr) \cdot k^{-1} \ (mod\, q)$$
$$\sigma = r$$

Verify:

Compute:

$$u_1 = H(m) \cdot s^{-1} \ (mod\, q)$$
$$u_2 = r \cdot s^{-1} \ (mod\, q)$$

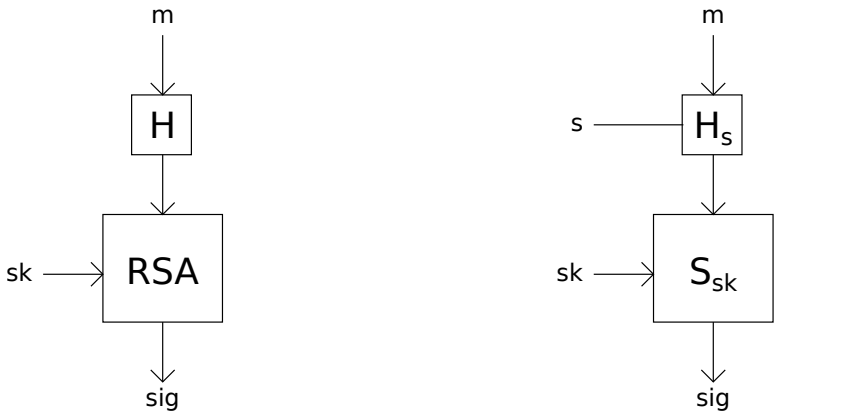
Check:

$$r = g^{u_1} \cdot y^{u_2} \ (mod\, p) \ (mod\, q)$$

- Widely used
- No security proof exists yet

3.3.4 Hash and Sign paradigm

- Hash the message before it is signed
- Hashed RSA proven to be existentially unforgeable
- Attacks now depend on collisions in hash values produced by H



(a) Hash and Sign for RSA

(b) General Hash and Sign paradigm

Figure 21

General Hash and Sign paradigm:

1 Hash with a random key s
Key can either be a standardised key known to everyone or attached to the signature body

2 Sign message with a signature function using shared key sk

3.4 Zero Knowledge Proofs

Problem: How to prove knowledge of a secret without disclosing any information about the secret.

Properties:

1 If the prover knows the secret then the proof always succeeds

2 There should exist a knowledge extractor M that can extract the secret knowledge from the prover in polynomial time

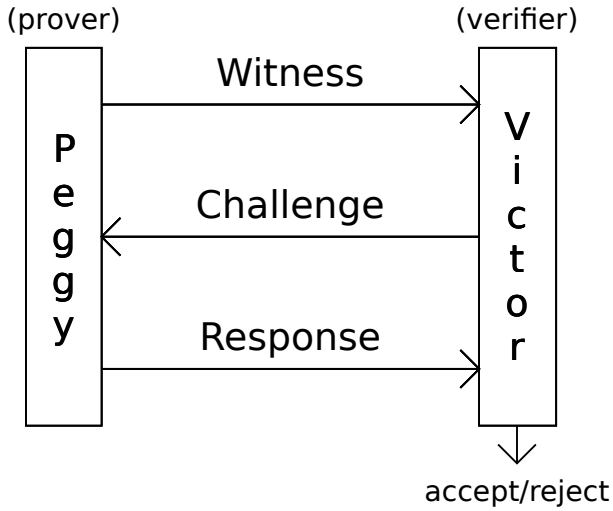


Figure 22: Interactive proof

3.4.1 Simulator

Generate a transcript (witness, challenge and response) without contacting a prover.

Using a simulator it is possible to generate a transcript that is indistinguishable from the transcript between a prover and a verifier.

Simulator computes transcript in reverse:

1 Choose random response

2 Choose a challenge

3 Compute the initial witness randomness

Possible because simulator is not bound to initial witness.

3.4.2 Schnorr protocol

Key generation: $GenSchnorr(1^n)$ (same as $GenDSA$, see 3.3.3).

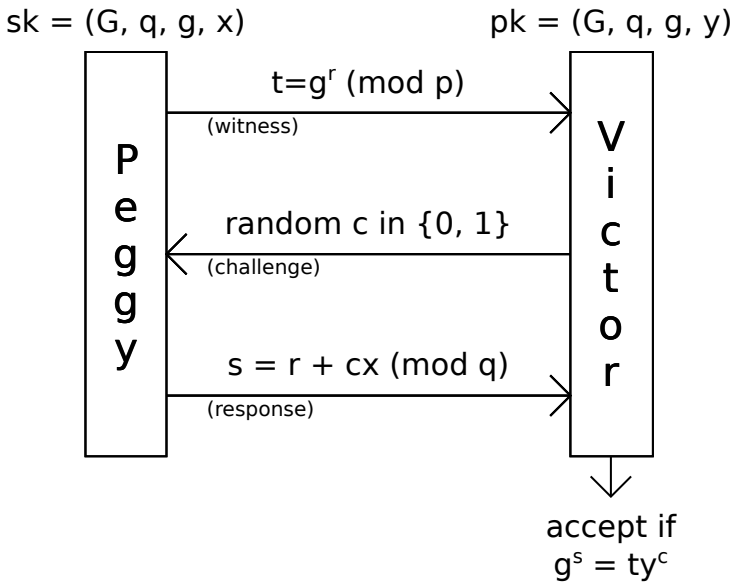


Figure 23: Schnorr protocol

Simulator for zero-knowledge proof:

1 Choose $c, s \in \mathbb{Z}_q$

2 Compute $t = g^s/y^c$

3 Transcript: (t, c, s)

Schnorr is honest verifier zero-knowledge.