Introduction to cryptography

Ragni Ryvold Arnesen
Ragni.Ryvold.Arnesen@nr.no
Norsk Regnesentral

Norsk Regnesentral
Norwegian Computing Center

Contents

- Security characteristics
- Symmetric crypto algorithms
  - Stream ciphers
  - Block ciphers
- Asymmetric crypto algorithms
  - Factorisation problem
  - RSA
  - Hashing
  - Digital signatures
  - ElGamal

Terminology

- *P* is a finite set of possible *plaintexts*
- *C* is a finite set of possible *cryptotexts*
- *K* is a finite set of possible *keys* (*keyspace*)

- For each $k \in K$ there is an *encryption function* $e_k: P \rightarrow C$, and a corresponding *decryption function* $d_k: C \rightarrow P$ such that $d_k(e_k(x))=x$ for every plaintext $x \in P$

Security characteristics

- Perfect Secrecy (or *unconditional* security):
  - The system is unbreakable even with infinite computational resources
- Computational Security:
  - The perceived level of computation required to break the security exceeds, by a comfortable margin, the computational resources of the adversary

Perfect secrecy

- A cryptosystem has *perfect secrecy* if $p_P(x|y) = p_P(x)$ for all $x \in P$
- In other words: The *a posteriori* probability that the plaintext is *x*, given that the ciphertext *y* is observed, is identical to the *a priori* probability that the plaintext is *x*
- It follows that not even exhaustive search through the entire keyspace will give any knowledge of the plaintext or the key
- Disadvantage: The amount of key needed is at least as big as the amount of plaintext

One-time pad

- The *one-time pad* is the only known cryptoalgorithm that achieves perfect secrecy

- Let $P = C = K = (\mathbb{Z}_2)^n$,
  - plaintext $x = (x_1, x_2, x_3, \ldots, x_n)$,
  - key $k = (k_1, k_2, k_3, \ldots, k_n)$, must be truly random!
  - cryptotext $y = (y_1, y_2, y_3, \ldots, y_n)$

Encryption:

$$e_k(x) = (x_1 \oplus k_1,\ x_2 \oplus k_2,\ x_3 \oplus k_3, \ldots,\ x_n \oplus k_n)$$

Decryption:

$$d_k(y) = (y_1 \oplus k_1,\ y_2 \oplus k_2,\ y_3 \oplus k_3, \ldots,\ y_n \oplus k_n)$$

Confusion and diffusion

- A good algorithm should ensure a high level of confusion and diffusion.

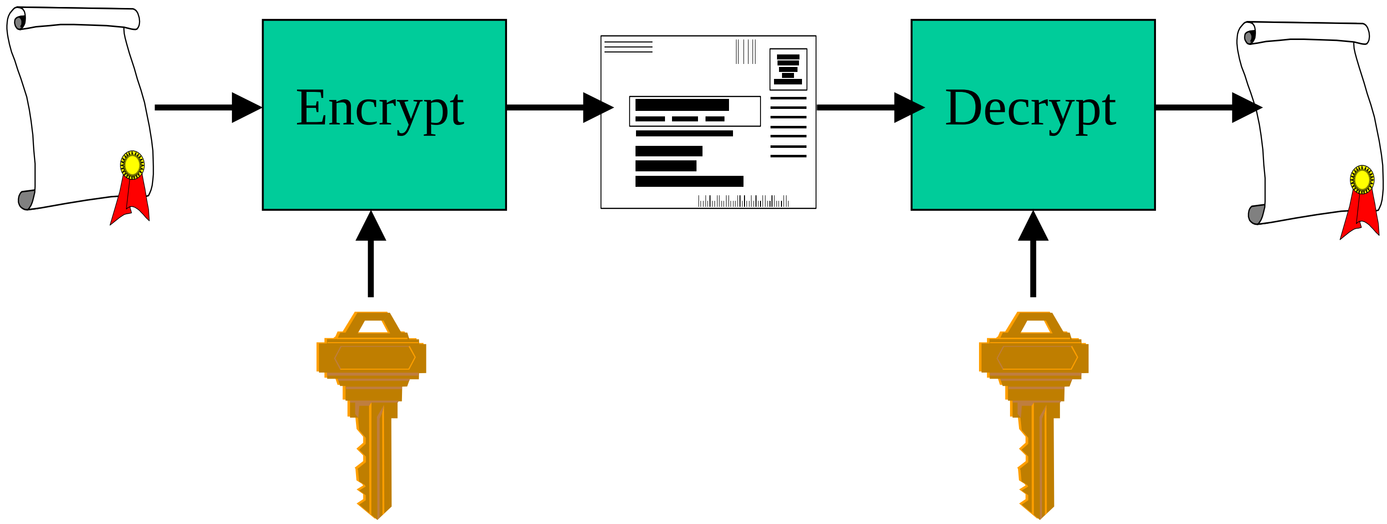
Confusion:

- Relationship between key and ciphertext is as complex as possible.
- One bit change in the key should result in change in approximately half of the ciphertext bits.

Diffusion:

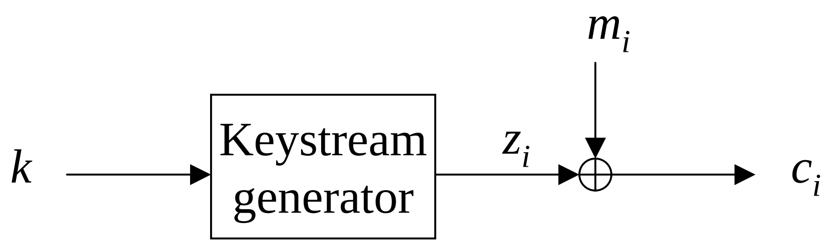
- Redundancy of the plaintext is spread out over the ciphertext.
- One bit change in the plaintext should result in change in approximately half of the ciphertext bits.

Symmetric crypto algorithms



- The same key is used for encryption and decryption.
- The keys must be secret and shared in advance (off-line or by some key exchange mechanism)
- Symmetric cryptoalgorithms are used mainly to ensure
  - Confidentiality (conceal contents of data)
  - Integrity (protect data from change)

Stream ciphers



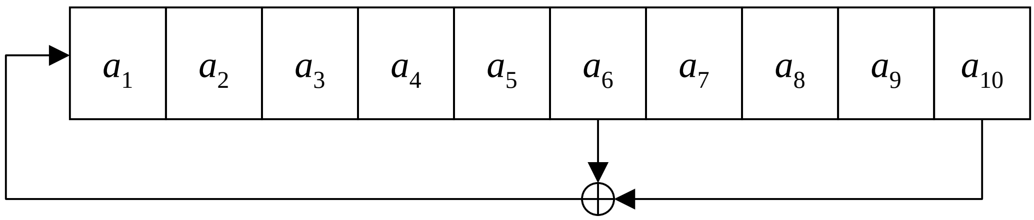
plaintext m_i
ciphertext c_i
key k
keystream z_i

Properties of a stream cipher:

- encrypts individual characters, one at a time
- the encryption transformation varies with time
- usually fast and simple in hardware
- no need for buffering plaintext or cryptotext
- limited or no error propagation
- much of the theory dates back to around World War II and is extensively analysed
- few algorithms published in the open literature
- widely used in telecommunications, radios and military communication equipment

LFSR -
Linear Feedback Shift Register

State polynomial: $a_1 x^9 + a_2 x^8 + a_3 x^7 + a_4 x^6 + a_5 x^5 + a_6 x^4 + a_7 x^3 + a_8 x^2 + a_9 x + a_{10}$



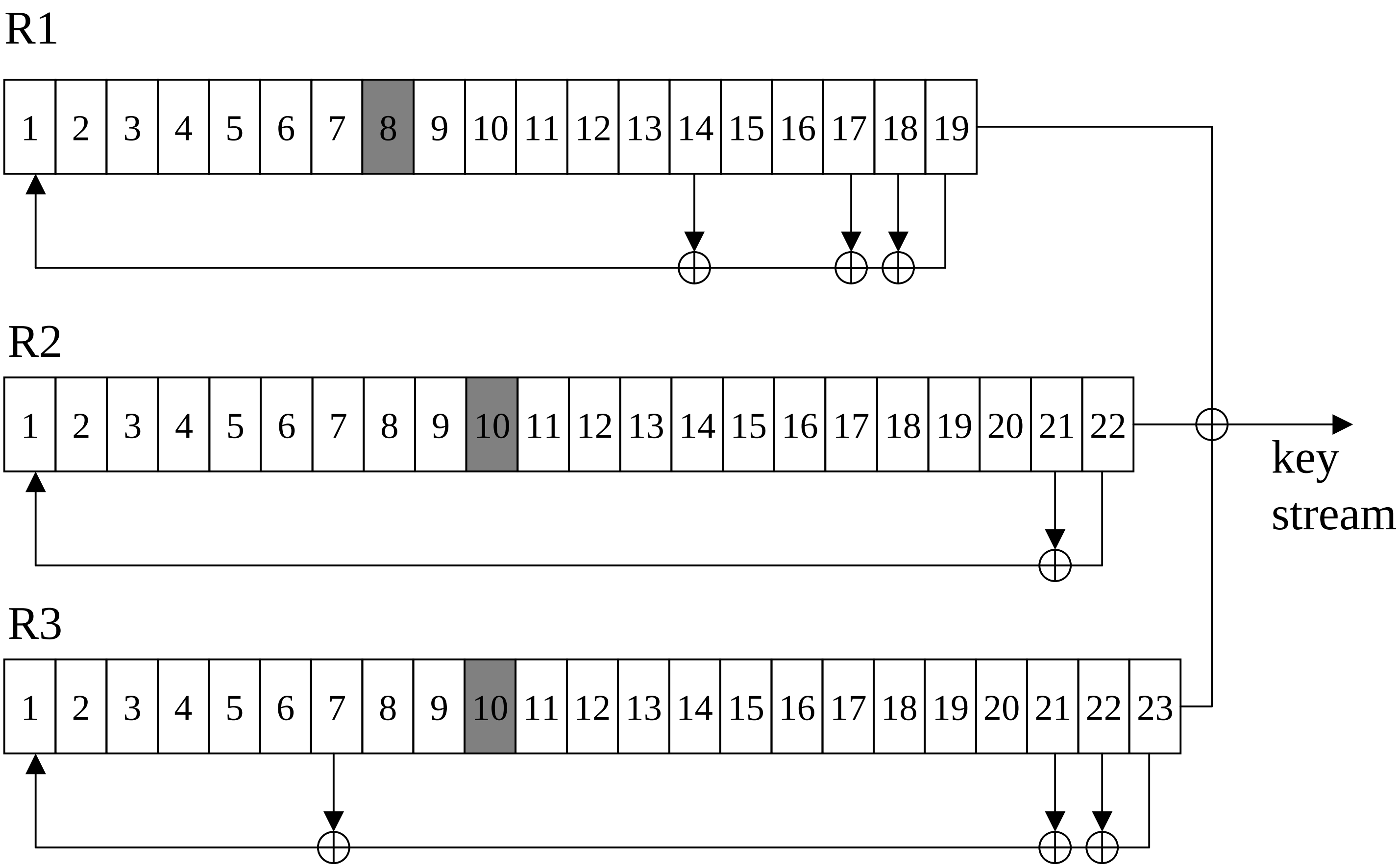
- Corresponds to the *connection polynomial*

$$x^{10} + x^6 + 1$$

- If the polynomial is *primitive*, the LFSR will have its maximum possible *period* 2^n-1, where n is the length of the LFSR
- Stepping the LFSR once corresponds to multiplying the *state polynomial* with x and reducing modulo the *connection polynomial*

- LFSRs are very often used as parts of a stream cipher

GSM cipher - A5/1

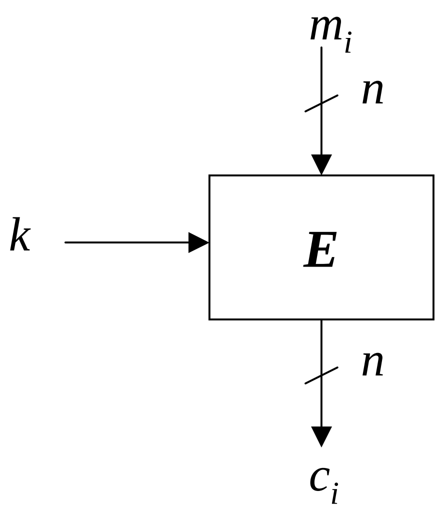


- A register is *clocked* if its *clocking tap* (marked grey) agrees with the majority of the three clocking taps.

Cryptanalysis of A5/1

- 64-bit keys, but in all implementations 10 bits are set to zero
- Anderson and Roe, 1994
  - Guess R1 and R2 (41 bits) and derive R3 from the output, complexity about $O(2^{45})$
- Time/memory trade-off (Babbage 1995, Golic 1997)
  - Complexity $O(2^{22})$ with 64TB diskspace, or
  - Complexity $O(2^{28})$ with 862GB diskspace
- Best attack known : Alex Biryukov, Adi Shamir and David Wagner, 1999-2000
  - Preparation: 2^{48} (carried out only once)
  - 2 min known plaintext: key computed in 1 sec.
  - 2 sec known plaintext: key computed in a few minutes
  - Question: How to get hold of the plaintext?

Block ciphers



encryption function E
plaintext m_i
ciphertext c_i
key k

Properties of a block cipher:

- maps n-bit plaintext blocks to n-bit ciphertext blocks
- pure block ciphers are *memoryless*
- many algorithms in the open literature that have been extensively analysed (DES, IDEA, AES, etc.)
- widely used in e-commerce and banking

Norsk Regnesentral
Norwegian Computing Center

UMTS cipher - KASUMI

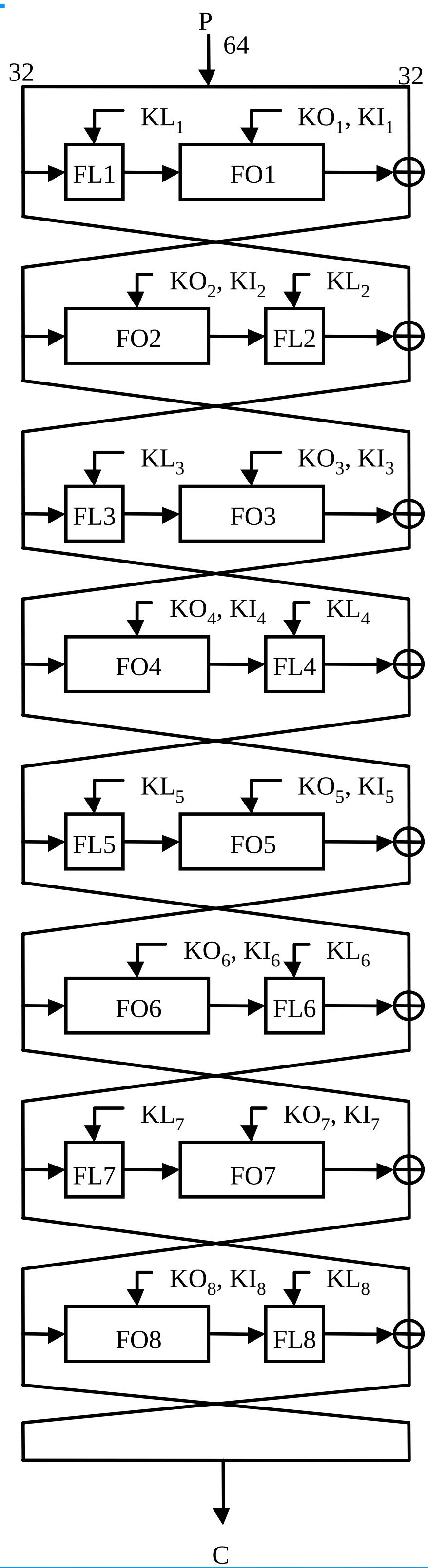


Fig. 1: Modified MISTY1

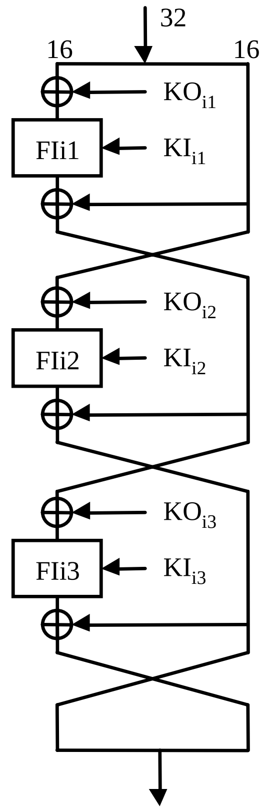


Fig.2: FO Function

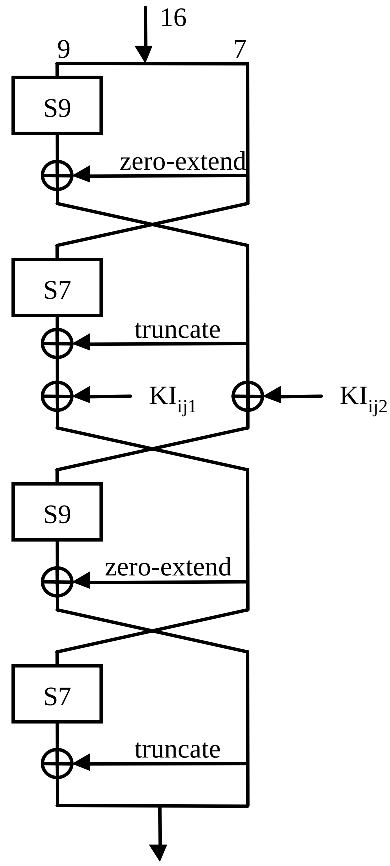


Fig.3: FI Function

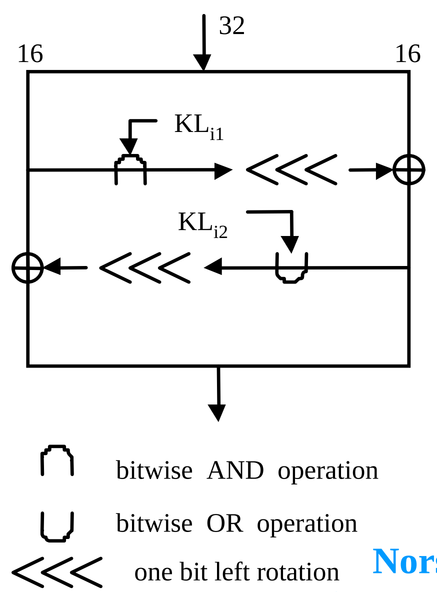


Fig.4: FL Function

Norsk Regnesentral
Norwegian Computing Center

S-boxes: S7

Input: $(x_6, x_5, x_4, x_3, x_2, x_1, x_0)$

Output: $(y_6, y_5, y_4, y_3, y_2, y_1, y_0)$

Gate Logic:

$$y_0 = x_1x_3 + x_4 + x_0x_1x_4 + x_5 + x_2x_5 + x_3x_4x_5 + x_6 + x_0x_6 + x_1x_6 + x_3x_6 + x_2x_4x_6 + x_1x_5x_6 + x_4x_5x_6$$

$$y_1 = x_0x_1 + x_0x_4 + x_2x_4 + x_5 + x_1x_2x_5 + x_0x_3x_5 + x_6 + x_0x_2x_6 + x_3x_6 + x_4x_5x_6 + 1$$

$$y_2 = x_0 + x_0x_3 + x_2x_3 + x_1x_2x_4 + x_0x_3x_4 + x_1x_5 + x_0x_2x_5 + x_0x_6 + x_0x_1x_6 + x_2x_6 + x_4x_6 + 1$$

$$y_3 = x_1 + x_0x_1x_2 + x_1x_4 + x_3x_4 + x_0x_5 + x_0x_1x_5 + x_2x_3x_5 + x_1x_4x_5 + x_2x_6 + x_1x_3x_6$$

$$y_4 = x_0x_2 + x_3 + x_1x_3 + x_1x_4 + x_0x_1x_4 + x_2x_3x_4 + x_0x_5 + x_1x_3x_5 + x_0x_4x_5 + x_1x_6 + x_3x_6 + x_0x_3x_6 + x_5x_6 + 1$$

$$y_5 = x_2 + x_0x_2 + x_0x_3 + x_1x_2x_3 + x_0x_2x_4 + x_0x_5 + x_2x_5 + x_4x_5 + x_1x_6 + x_1x_2x_6 + x_0x_3x_6 + x_3x_4x_6 + x_2x_5x_6 + 1$$

$$y_6 = x_1x_2 + x_0x_1x_3 + x_0x_4 + x_1x_5 + x_3x_5 + x_6 + x_0x_1x_6 + x_2x_3x_6 + x_1x_4x_6 + x_0x_5x_6$$

Decimal Table:

54	50	62	56	22	34	94	96	38	6	63	93	2	18	123	33
55	113	39	114	21	67	65	12	47	73	46	27	25	111	124	81
53	9	121	79	52	60	58	48	101	127	40	120	104	70	71	43
20	122	72	61	23	109	13	100	77	1	16	7	82	10	105	98
117	116	76	11	89	106	0	125	118	99	86	69	30	57	126	87
112	51	17	5	95	14	90	84	91	8	35	103	32	97	28	66
102	31	26	45	75	4	85	92	37	74	80	49	68	29	115	44
64	107	108	24	110	83	36	78	42	19	15	41	88	119	59	3

S9 is constructed similarly, but with $2^9 = 512$ entries in the table.

Norsk Regnesentral
Norwegian Computing Center

Key schedule

Secret Key

K	128 bit	

Subkey

Ki (1 <= i <= 8)	16 bit	K = K1 || K2 || K3 || ...... || K8
Ki' (1 <= i <= 8)	16 bit	Ki' = Ki XOR Ci

Key Symbols

KLi (1 <= i <= 8)	32 bit	KLi = KLi1 || KLi2
KLij (1 <= i <= 8) (1 <= j <= 2)	16 bit	
KOi (1 <= i <= 8)	48 bit	KOi = KOi1 || KOi2 || KOi3
KOij (1 <= i <= 8) (1 <= j <= 3)	16 bit	
KIi (1 <= i <= 8)	48 bit	KIi = KIi1 || KIi2 || KIi3
KIij (1 <= i <= 8) (1 <= j <= 3)	16 bit	KIi = KIij1 || KIij2
KIij1 (1 <= i <= 8) (1 <= j <= 3)	9 bit	
KIij2 (1 <= i <= 8) (1 <= j <= 3)	7 bit	

Subkey – KeySymbol Relation

	i = 1	i = 2	i = 3	i = 4	i = 5	i = 6	i = 7	i = 8
KLi1	K1<<<1	K2<<<1	K3<<<1	K4<<<1	K5<<<1	K6<<<1	K7<<<1	K8<<<1
KLi2	K3'	K4'	K5'	K6'	K7'	K8'	K1'	K2'
KOi1	K2<<<5	K3<<<5	K4<<<5	K5<<<5	K6<<<5	K7<<<5	K8<<<5	K1<<<5
KOi2	K6<<<8	K7<<<8	K8<<<8	K1<<<8	K2<<<8	K3<<<8	K4<<<8	K5<<<8
KOi3	K7<<<13	K7<<<13	K7<<<13	K7<<<13	K7<<<13	K7<<<13	K7<<<13	K7<<<13
KIi1	K5'	K6'	K7'	K8'	K1'	K2'	K3'	K4'
KIi2	K4'	K5'	K6'	K7'	K8'	K1'	K2'	K3'
KIi3	K8'	K1'	K2'	K3'	K4'	K5'	K6'	K7'

Constant Values

C1 = 0x0123
C2 = 0x4567
C3 = 0x89ab
C4 = 0xcdef
C5 = 0xfedc
C6 = 0xba98
C7 = 0x7654
C8 = 0x3210

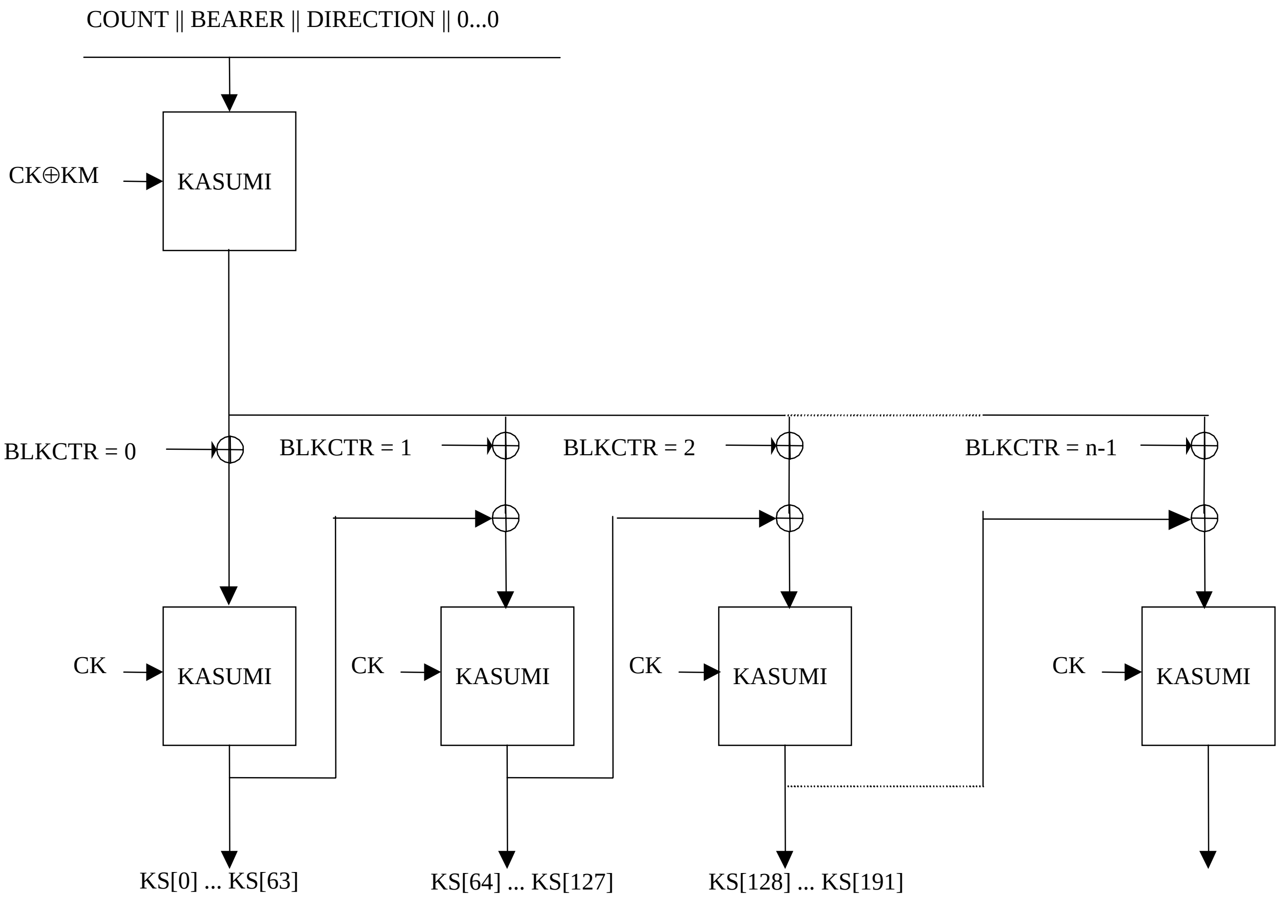
Modes of use

- A block cipher is seldom used in its pure form (*n* bits plaintext in, *n* bits plaintext out)
- Instead it is used in one of several possible *modes* depending on the objectives:
  - Confidentiality protection
  - Integrity protection
  - Key generation
  - Key exchange
  - Challenge-response protocol
  - etc.

UMTS Confidentiality algorithm - f8

Parameters

COUNT	32 bits	time dependent input
BEARER	5 bits	bearer identity
DIRECTION	1 bit	direction of transmission
BLKCTR	64 bits	block counter
LENGTH	? bits	length of key stream
CK	128 bits	cipher key
$\{PT_i\}_{i=0,1,1,\ldots,LENGTH-1}$		plaintext bit sequence
$\{CT_i\}_{i=0,1,1,\ldots,LENGTH-1}$		ciphertext bit sequence
$\{KS_i\}_{i=0,1,1,\ldots,LENGTH-1}$		output key stream



CT[i] = PT[i] XOR KS[i]

Norsk Regnesentral
Norwegian Computing Center

Message Authentication Code (MAC)

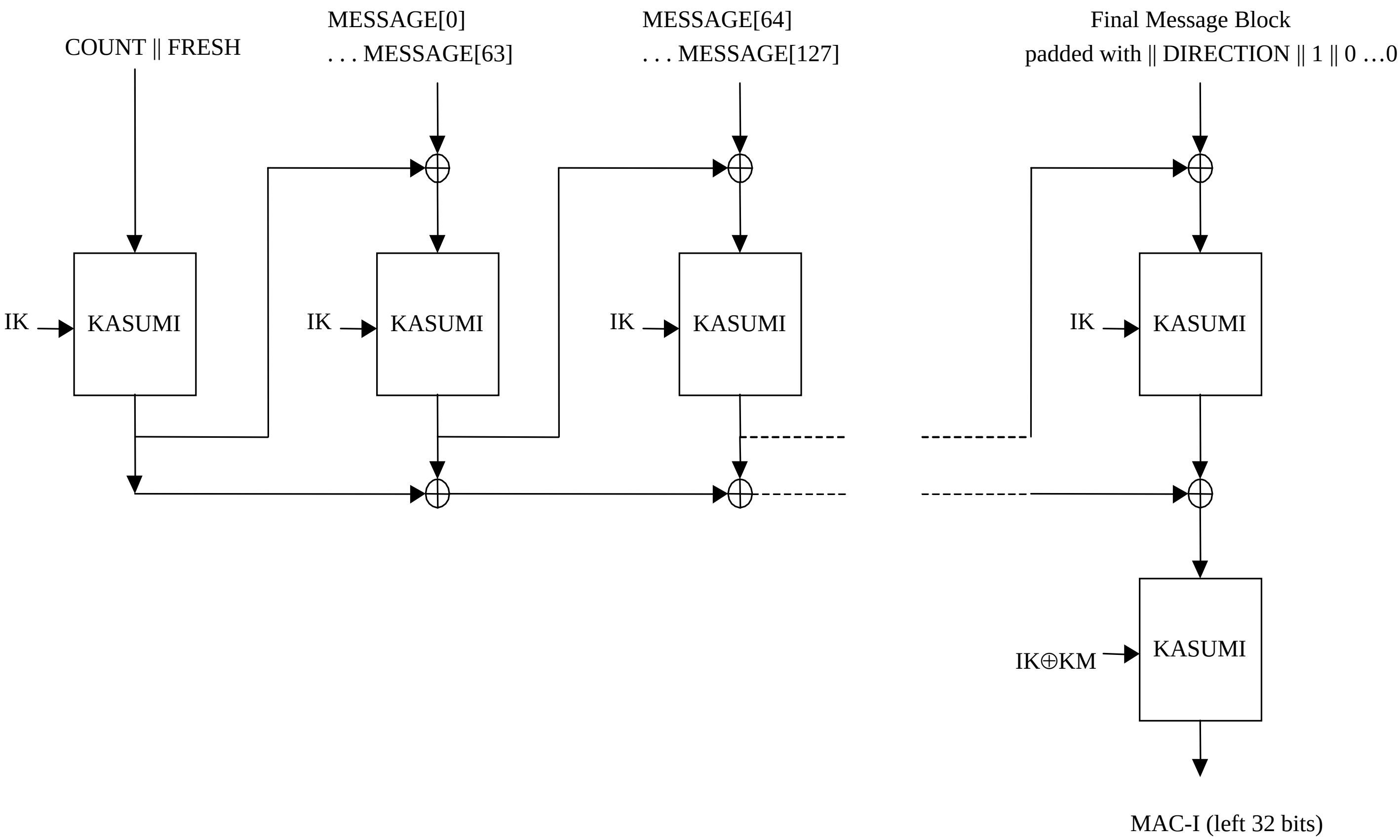
- Used to ensure *integrity* of data
- Maps an arbitrary-length message onto a fixed-length output (MAC)
  - Key dependent
  - Often based on a block-cipher
- The MAC is attached to the cryptotext, and by verifying it, the receiver knows two things:
  - the message was produced by the someone holding the secret integrity key
  - the message has not been changed during transmission

Norsk Regnesentral
Norwegian Computing Center

UMTS Integrity algorithm - f9

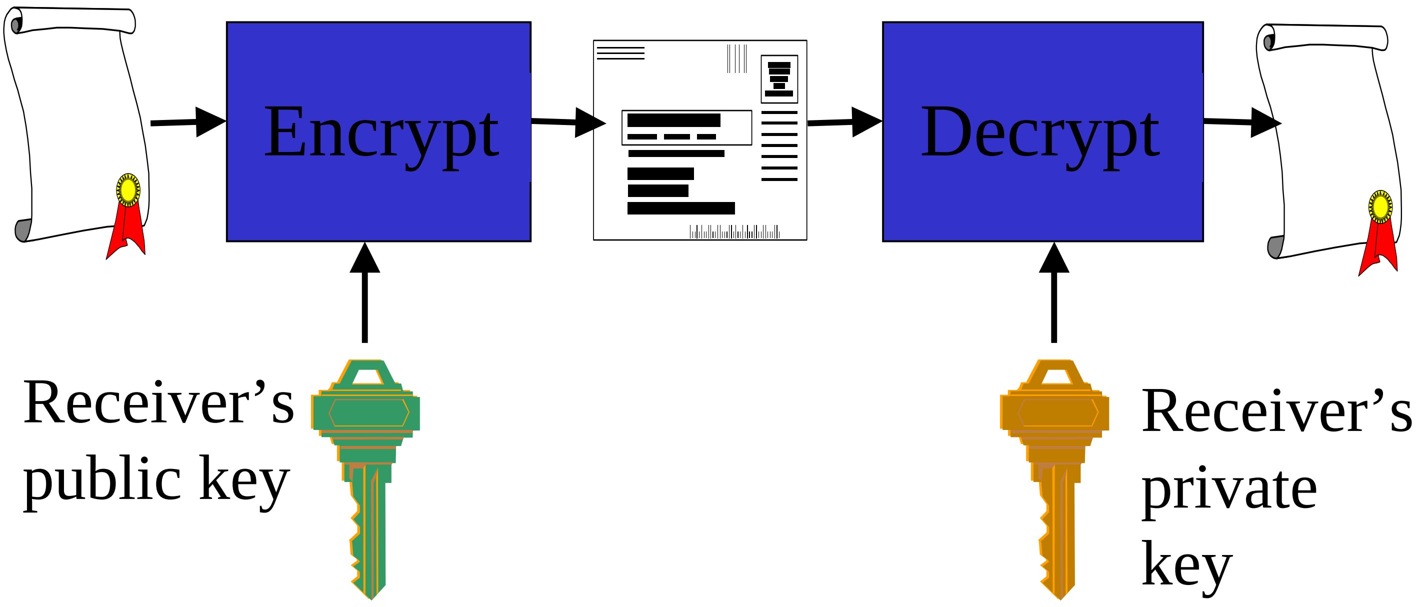
Parameters

COUNT	32 bits	time dependent input
FRESH	32 bits	random number
DIRECTION	1 bit	direction of transmission
IK	128 bits	integrity key
$\{MESSAGE\}_{i=0,1,1,\ldots,LENGTH-1}$		plaintext bit sequence
MAC-I	32 bits	message authentication code



Norsk Regnesentral
Norwegian Computing Center

Asymmetric (public key) crypto algorithms



- Encrypt with *receiver's* public key
- Receiver decrypts with his private key
- *N* public keys for *N* parties (as opposed to *N*(*N*-1) for symmetric cryptosystems)

Services

- Confidentiality
  - Conceal contents of data
- Integrity
  - Detect change of data
- Authentication
  - Establish identity of communicating parties
  - Establish identity of data origin
- Non-repudiation
  - Convince third party that an action
    - has been executed by a certain individual
    - has been executed at a given point in time

The integer factorisation problem

- Given a positive integer n, find its prime factorisation, i.e. write $n = p_1^{e1} p_2^{e2} \ldots p_k^{ek}$ where the p_i *are* pairwise distinct primes and each $e_i \geq 1$
- Factoring algorithms:
  - Trial division
  - Pollard rho method
  - Pollard's p -1 method
  - Quadratic sieve
  - Lenstra's elliptic curve method
  - Number field sieve

Norsk Regnesentral
Norwegian Computing Center

Number theory

- Definition:
  - Two positive integers x and y are *relatively prime* if they have no common factors, i.e. their greatest common divisor is 1.
    We write gcd(x, y) = 1.
- Euler phi function:
  - Let n be a positive integer. The Euler *phi function* $\varphi(n)$ is the number of positive integers not exceeding n that are relatively prime to n
- Theorem:
  - If p is prime, then $\varphi(p) = p - 1$
- Theorem:
  - Let m and n be relatively prime positive integers. Then $\varphi(mn) = \varphi(m)\,\varphi(n)$
- Euler's theorem:
  - If m is a positive integer and a is an integer with gcd(a,m) = 1, then $a^{\varphi(m)} \equiv 1 \pmod{m}$
- Fermat's theorem:
  - Special case of Euler's theorem: If gcd(a, p) = 1, then $a^{p-1} \equiv 1 \pmod{p}$

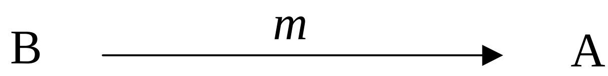
Norsk Regnesentral
Norwegian Computing Center

RSA - key generation

- Each entity A should do the following:
  - Generate large primes p and q
  - Compute $n = pq$ and $\varphi = (p - 1)(q - 1)$
  - Select random integer e, $1 < e < \varphi$, such that $\gcd(e, \varphi) = 1$
  - Compute the unique integer d, $1 < d < \varphi$, such that $ed \equiv 1 \pmod{\varphi}$
  - A's public key is (n, e), A's private key is d
- (Note that p, q and φ must also be kept secret)

- Conjecture:
  - Nobody can compute
    - p, q or φ from knowledge of n, or
    - d from knowledge of n and e

RSA - encryption



- Encryption. *B* should do the following:
  - Obtain *A*'s public key (n, e)
  - Represent the message as an integer m in the interval [0, n -1]
  - Compute $c = m^e \bmod n$
  - Send the ciphertext c to *A*

- Decryption. *A* should do the following
  - Use the private key d to recover $m = c^d \bmod n$

NR

RSA - proof that decryption works

- $ed \equiv 1 \pmod{\varphi} \Rightarrow$ there exists integer k such that $ed = 1+k\varphi$
- By Euler's theorem: $m^{\varphi} \equiv 1 \pmod{n}$
  - (This is true only if gcd(m,n) = 1. But if not, then we have found a factor of n, and the key is broken! The probability for this is extremely small.)

$\Rightarrow m^{k\varphi} \equiv 1 \pmod{n}$

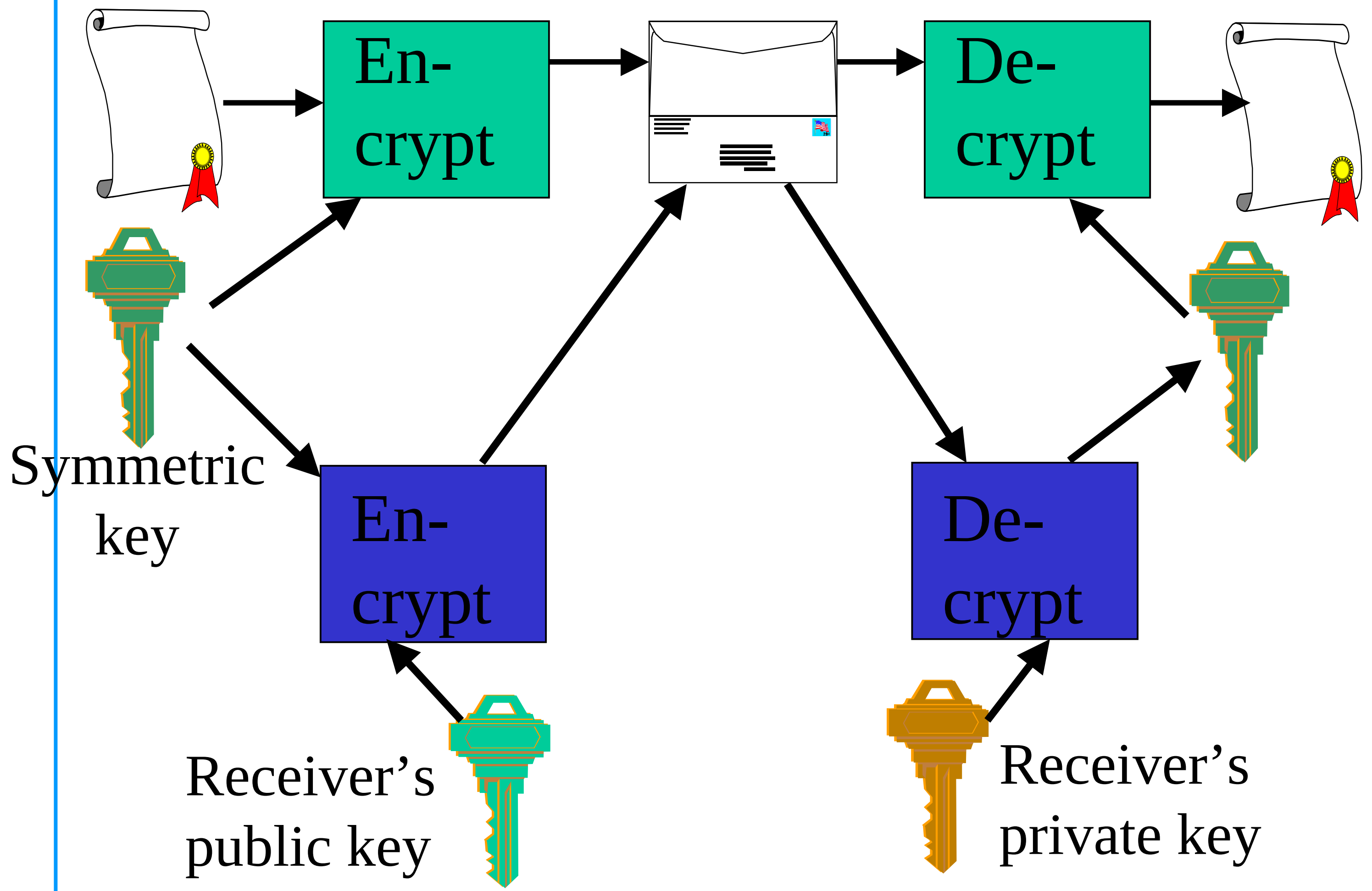
$\Rightarrow m^{k\varphi+1} \equiv m \pmod{n}$

$\Rightarrow m^{ed} \equiv m \pmod{n}$

$\Rightarrow c^{d} = (m^{e})^{d} = m^{ed} \equiv m \pmod{n}$

Norsk Regnesentral
Norwegian Computing Center

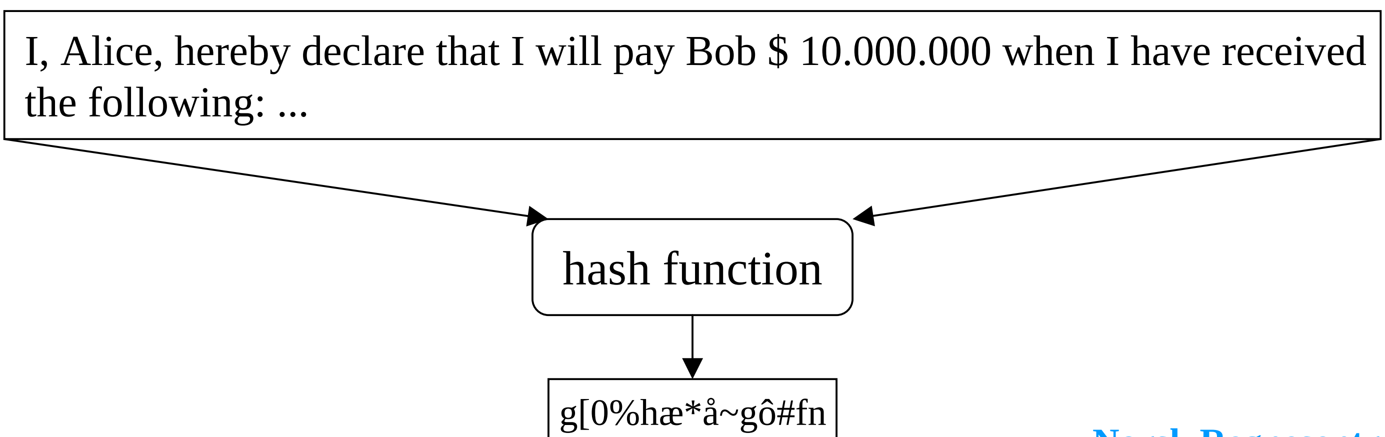
Hybrid method



- Public key is used to encrypt symmetric key

Hashing

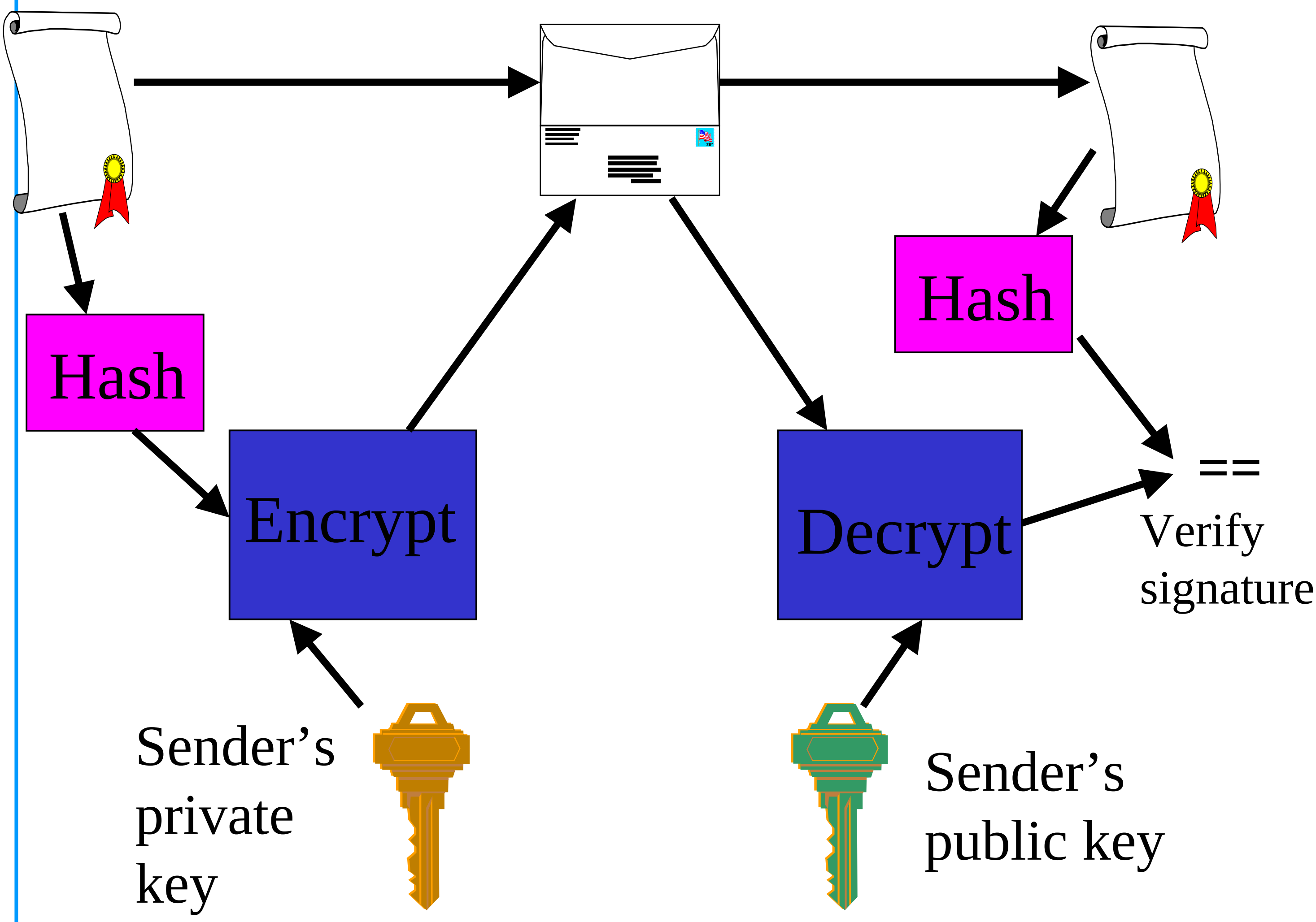
- One-way function:
  - A function *f* such that *f*(*x*) is easy to compute for each *x* in the domain of *f* ; but it is computationally infeasible to find any *x* such that *f*(*x*) = *y*, for essentially all *y* in the range of *f*
    - It is not known whether real one-way functions exist
- Hash function
  - A one-way function where variable-length input is mapped to fixed-length output



Security properties for hash functions

- Let h be a hash function with inputs x, x' and outputs y, y'.
- Preimage resistance (or *one-way*):
  - For essentially all pre-specified outputs y, it is computationally infeasible to find any preimage x' such that $h(x') = y$
- 2nd preimage resistance (or *weak* collision resistance):
  - Given x, it is computationally infeasible to find any $x' \neq x$ such that $h(x) = h(x')$
- Collision resistance (*strong* c.r.):
  - It is computationally infeasible to find any two distinct inputs x, x' such that $h(x) = h(x')$

Digital signatures



- Sign with sender's *private* key
- Verify signature with public key

Digital signatures

- When the receiver has verified the signature he knows that:
  - the document is really written by the person who owns the public key, i.e. the person who knows the corresponding private key (authentication of data origin)
  - the document has not been changed after the sender signed it since the hashes match (integrity of data)
- And:
  - The receiver can convince a *third party* that the contents of the document was really written by the sender (non-repudiation)

RSA signature

- Key generation as for encryption
- Signature generation. *A* should do the following:
  - if *M* is the message, compute $m = h(M)$, an integer in the range $[0, n-1]$
  - compute $s = m^d \bmod n$
  - *A*'s signature for *M* is *s*
- Verification. *B* should:
  - obtain *A*'s public key (n, e)
  - compute $m' = s^e \bmod n$ and $h(M)$
  - verify that $m' = h(M)$
- ($h()$ is a hash function)

Discrete logarithm problem (DLP)

- The *generalised discrete logarithm problem* is the following:
  - Given a finite cyclic group G of order n, a generator α of G, and an element $\beta \in G$, find the integer x, $0 \le x \le n-1$, such that $\alpha^x = \beta$
- Algorithms for solving the DLP:
  - Exhaustive search
  - Baby-step giant-step
  - Pollard's rho algorithm
  - Pohlig-Hellman algorithm
  - Index calculus algorithms

ElGamal - key generation

- Each entity *A* should do the following:
  - Generate a large random prime *p* and a generator *α* of the multiplicative group $\mathbb{Z}_p^*$
  - Select random integer *a* such that $1 \leq a \leq p - 2$
  - Compute $y = \alpha^a \bmod p$
  - *A*'s public key is (p, α, y), *A*'s private key is *a*

- Conjecture:
  - Nobody can compute *a* from knowledge of *y* and *α*

Norsk Regnesentral
Norwegian Computing Center

ElGamal - signature

- Signature generation. *A* should do the following:
  - Select random secret integer k, $1 < k < p - 2$ with $\gcd(k, p - 1) = 1$
  - Compute $r = \alpha^k \bmod p$
  - Compute $k^{-1} \bmod (p - 1)$
  - Compute $s = k^{-1}(h(m) - ar) \bmod (p - 1)$
  - *A*'s signature for *m* is the pair (r, s)

- Verification. *B* should:
  - Obtain *A*'s authentic public key (p, α, y)
  - Verify that $1 \leq r \leq p - 1$; if not, reject signature
  - Compute $v_1 = y^r r^s \bmod p$
  - Compute $h(m)$ and $v_2 = \alpha^{h(m)} \bmod p$
  - Accept the signature if and only if $v_1 = v_2$

($h()$ is a hash function)

ElGamal - proof that signature verification works

- Assume (r, s) is a legitimate signature of entity A on message m

$\Rightarrow s \equiv k^{-1}(h(m) - ar) \pmod{p-1}$ (1)

$\Rightarrow h(m) \equiv ar + ks \pmod{p-1}$ (2)

$\Rightarrow \alpha^{h(m)} \equiv \alpha^{ar+ks} \equiv (\alpha^a)^r\, r^s \pmod{p}$ (3)

$\Rightarrow v_2 = v_1$

- Between (2) and (3):
  - Theorem: Let a, n be relatively prime integers and $n > 0$. Then $a^i \equiv a^j \pmod{n}$ where i and j are positive integers, if and only if $i \equiv j \pmod{ord_n\, a}$.
  - Here, $ord_n\, a$ is the least positive integer x such that $a^x \equiv 1 \pmod{n}$, so if a is a generator of $\mathbf{Z}_p^*$ then $ord_n\, a = p - 1$

Norsk Regnesentral
Norwegian Computing Center