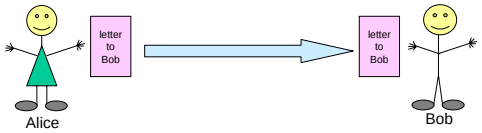
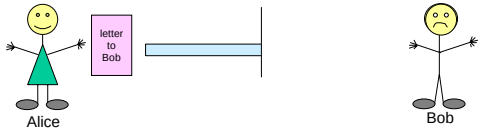
5 Cryptographic Algorithms

How can we design “secure communication”?

- **"normal" communication:**

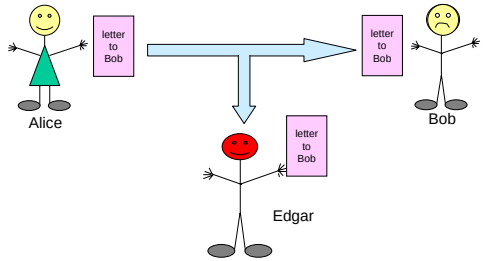


- **Security problems:**
  - **disruption:**



Bob does not get information that Alice has sent him a message.

passive attack:

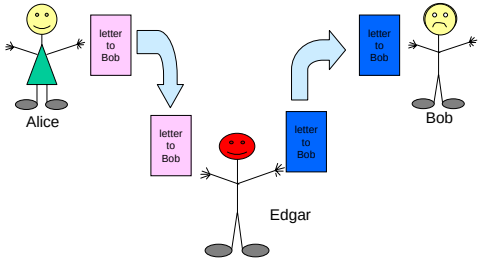


enemy is listening

- Edgar is tapping on the transmission channel and thereby can
  - read the messages exchanged between Alice and Bob (unless encryption is used)
  - analyse the complete data flow between Alice and Bob

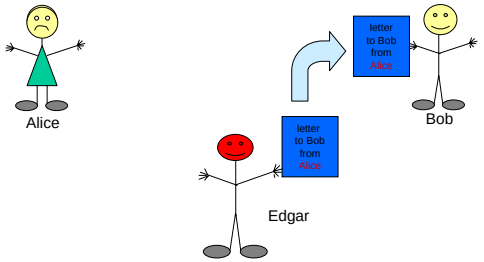
active attacks:

- **redirection:**



- Edgar redirects the data flow between Alice and Bob and thereby can
  - delay message transmission or change the sequential ordering of messages
  - modify the contents of messages

Masquerade:



- Edgar generates his own messages which he sends to Bob claiming to be “Alice”.

Requirements of secure communication:

- **Protection against passive attacks** by
  - **access control**: no unauthorised person may read file contents or get access to transmission channels
  - **encryption**: no unauthorised person may retrieve the information contained in the message or file (rsp. only at a cost which by far exceeds the value of the information).
- **Protection against active attacks** by
  - **authentication**: correct identification of the sender of a message
  - **integrity check**: guarantee that the data and data flow have not been changed (i.e. correct sender, contents, ordering, and time stamps)

- Nowadays, our private and professional activities rely heavily on the secure use of computers for storage, retrieval, and exchange of information.
- Essential means of achieving secure communication is the use of encryption.

⇒ ***Today, management of information infrastructures requires knowledge of cryptographic methods.***

The IFIP Position on Cryptopolicies

- **Preface:**

a. IFIP recognises the **highly important role of cryptographic mechanisms. In the Global Information Infrastructure (GII) and in Electronic Commerce** these mechanism will influence acceptability, usage, and competitiveness.

b. IFIP takes notice that for the convenience of discussion it is helpful to **distinguish between the differing objectives for the use of cryptographic mechanisms** - preservation of confidentiality, provision of the ability to authenticate people/organisations, provision of the ability to prove the integrity/completeness of data, etc.

c. IFIP is fully convinced that a **range of cryptographic mechanisms is required** to meet the security needs of the GII. Users may select the most effective for their specific purposes.

d. IFIP recognises that cryptography at the same time is prone to **potential abuse by criminals**. In this context law enforcement plays also an important role and we face the situation that different countries exhibit different attitudes.

e. Being aware that responsibilities for crime prevention and detection lies at national governments and that business is less and less related to national borders IFIP recognizes that **cryptographic services and cryptographic applications cannot be bound to a nation's territory**.

f. IFIP recognises the technical consensus that forbidding or **restricting the use of strong cryptography is from a technical standpoint ultimately unfeasible**. Taking the above said into account IFIP takes the following position on the use and regulation of cryptographic methods :

IFIP position on the use and regulation of cryptography

I. Cryptography has equal impact and importance when data are stored or transmitted. A distinction is unrealistic in a world of networked computers.

II. It is the prime goal that, whoever is involved in the process, cryptographic procedures and keys are handled in a way that full confidence of all partners, including the public at large, is assured.

III. It is desirable that voluntary and free use be in place for all types of cryptography.

IV. While a business will generally take precautions to protect itself against lost/forgotten/stolen keys, such considerations should be carefully separated from the law enforcement considerations, even though the mechanisms for each may be the same or overlap.

V. When establishing key management and cryptography infrastructures this should be primarily driven by the users needs and not by regulatory requirements.

VI. Law enforcement shall not establish methods in the cryptography context that infringe on a citizen's expectations of personal privacy and integrity within a country.

VII. IFIP assumes that organised and major crime will successfully avoid or evade any requirement to comply with a key deposit scheme. Law enforcers must therefore not rely primarily on key deposit schemes when addressing the issue of criminal intelligence gathering. Research should be conducted, which results in a set of appropriate, acceptable, and well focused alternative methods.

VIII. In cases where keys are deposited at third parties it is necessary that commercial and privacy interest as well as commercial liabilities must be guaranteed in all phases. This is particularly necessary if such systems allow law enforcement to access data in clear or keys, under proper legal constraint.

IX. There is a great need that cryptographic methods and especially digital signatures be recognised by national and international law. Such recognition carries with it responsibilities for assuring availability of relevant keys throughout any legally specified retention period and liabilities for improper disclosure of or change to keys whilst they are being kept.

X. Any legal or regulatory arrangement between two nations, in respect to cryptography and access to relevant materials, must be symmetric.

Cryptology

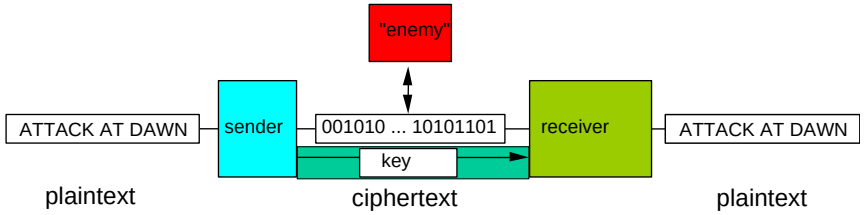
Investigation of systems for secret communication

- **cryptography** design of systems and methods for secret communication
- **cryptanalysis** investigation of methods for decryption

Goal : methods of cryptography have to make sure that the costs of cryptanalytic methods by far exceed the value of the encrypted information

- **in old times**: mainly military applications
- **nowadays**: general requirement for **privacy protection**

Typical cryptosystem



Assumptions

- Sender, receiver, and enemy know the encryption method.
- Sender and receiver use the same key parameters
  (e.g. the same key for symmetric methods or
  the appropriate pair of keys for asymmetric methods)
- The key parameters are hidden from the enemy.

Requirements:

- simple encryption and decryption
- the cost of "cracking" the code must be larger than the value of the plain text information

simple methods of encryption

Caesar Cipher:

- cyclic shift of the alphabet by k positions

example: (k=1)

- plaintext: A T T A C K A T D A W N
- cipher text: B U U B D L A B U A E B X O

- unreliable, since the key (value of k) is easily guessed.

Table-Cipher:

- replace letters according to a permutation (or encryption) of the alphabet

example:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
T H E Q U I C K B R O W N F X J M P D V G L A Z Y S

  - plain text: A T T A C K A T D A W N
  - cipher text: T D D T E R S T D S T L N

- easily decrypted by frequency analysis of letters and letter combinations

Vigenere Cipher:

- Use more than one table, e.g. a simple generalisation of the Caesar cipher with a short key determining cyclically the value of k for every letter.

example: key ABC (i.e. shift distances 1,2,3,1,2,3,...)

- key: A B C A B C A B C A B C A B
- plaintext: A T T A C K A T D A W N
- ciphertext: B V W B E N A C W A F D X P

One Time Pad:

- use a random key of the same length as the plaintext
- use key only once

⇒ only provably safe encryption technique, as long as the key is secret.
(said to have been used for the "hot wire" between Washington and Moscow)

Simple encryption method for bit sequences:

- generate random bit sequence of appropriate length (using a pseudo-random number generator, i.e. the receiver can generate the same sequence using the same generator with the same seed)
- form the bitwise XOR of the two sequences

⇒ the plain text can be retrieved from the ciphertext by another XOR with the same key.

  – Regularities of the plaintext are hidden by XORing with a random sequence.

```
plaintext:     01110011010111010001110101110111111011....
random seq.:   10110101100010010100001101101110101010 1....
ciphertext:    11000110110101000101111000011001011 10....
random seq.:   10110101100010010100001101101110101010 1....
plaintext:     01110011010111010001110101110111111011....
```

Similarities in all methods:

- character oriented **substitution**,
  (also **confusion** = hiding the relationship between plaintext and ciphertext characters)

Further cryptographic principle: Transposition

- **Transposition** = characters are not encrypted but their position in the text is changed.
  (special case of
  diffusion = distributing the information of the plaintext over the ciphertext)

example:

- write the plaintext into the columns of a matrix and read it out in row major order:

A	C	T	W
T	K		N
T		D	
A	A	A	

⇒ cipher text: A C T W T K N T D A A A

- In this case, the key is the form of the matrix rsp. the chosen permutation.

Further difference:

- Using substitution the characters are encrypted independently of each other, whereas here the text is considered as a whole, i.e. this is not a “stream cipher” (“Stromchiffrierung”) or "on-line" encryption.

Problems:

- Methods using substitution and/or transposition are quite easily attacked
  - exploiting frequency of letters and letter sequences
  - trying different transposition schemata
  - exploiting partial knowledge of the plaintext (greetings, dates, names,…)

Remedy:

- combine both approaches, and use methods with several stages of encryption.

One of the standard methods (although not an official standard anymore):

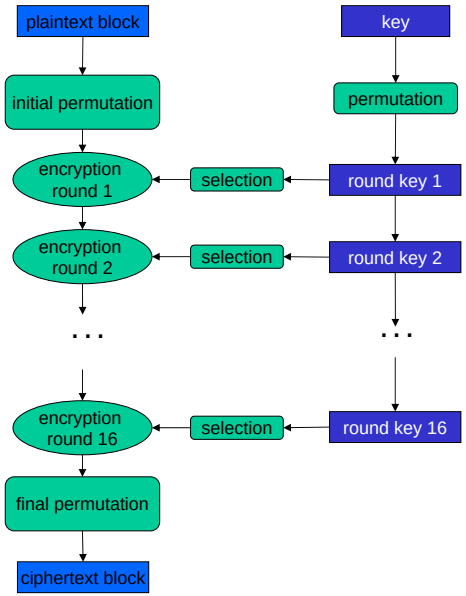
- **DES** = **D**ata **E**ncryption **S**tandard
  - developed by IBM in co-operation with NSA (National Security Agency)
  - in 1977 accepted as encryption standard, has recently been replaced by the Advanced Encryption Standard AES *(will be presented later)*
  - continuous encryption of plaintext blocks of 64 bits using a 56 bit key (“block cipher”)

Details of DES

- **abstract schema:**

- initial permutation = inverse final permutation

  (this permutation has no cryptographic relevance)



Key transformations:

- Permutation: no special meaning
- Round key is constructed by
  - dividing the key into two 28 bit parts
  - cyclic left shift of each half (by 1 bit in round 1, 2, 9 and 16, by 2 bits in the other ones)
  - selection (”compression permutation”) of 48 bits of the round key

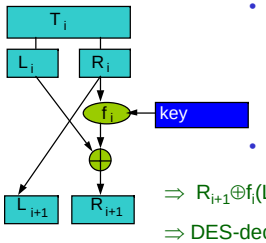
⇒ different bits of the key are used in every round; each one in about 14 rounds, but not in a regular way

- Every encryption round is a *"Feistel"-round* (named after a person from IBM):

T_i : text block before round i

L_i / R_i : left / right half of T_i

f_i : transformation

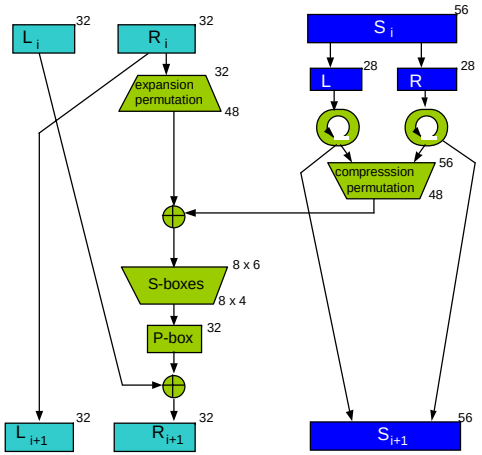


- Knowing L_{i+1}, R_{i+1}, f_i and the round key, one can retrieve L_i and R_i by simply applying another Feistel round:
- $R_{i+1} = L_i \oplus f_i(R_i)$, $L_{i+1} = R_i$

⇒ $R_{i+1} \oplus f_i(L_{i+1}) = L_i \oplus f_i(R_i) \oplus f_i(R_i) = L_i$

⇒ DES-decryption by another application of DES with the opposite round key sequence

Feistel round within DES :



- **compression permutation**: before called "selection"
- **expansion permutation**:
- 32 1 2 3 4 5 4 5 6 7 8 9 8 9 ... 27 28 29 28 29 30 31 32 1

⇒ blocks of 6 bits containing information from neighboring 4 bit blocks

S-Boxes:

- Each of the 8 S-boxes is a 4x16 table for transforming 6 bits into 4 bits:
  - 2 bits (first and last) row index, 4 bits column index, table entry is a 4bit value

example:

- S-Box No. 5

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

P-Box: Permutation, such that the output of each S-box will influence many other boxes in the next round.

Remarks:

- Expansion permutation and P-Box introduce diffusion (also called avalanche effect : every bit of the cipher text shall depend on every bit of the key and of the plain text)
- P-box manages that every plaintext bit passes through every S-Box on its "way through DES" .
- S-boxes introduce nonlinearity and immunity against "differential cryptanalysis".
- Rotation and selection make sure that every change of a key bit influences all cipher text bits after very few rounds.
- very well suited for hardware implementation (very common on smart cards).

cryptanalytic attacks on DES:

- **brute force:**
  Test all the $2^{56} \approx 72 \cdot 10^{15}$ possible keys.
  ⇒ computer with 1 Tera instr. per s. and 1 instr. per encryption needs 72.000 s = 20 h

  but: fastest DES-chips can do 25 to 250 Mill. encryptions per second.
  ⇒ even with 1 Giga enc./s one would need 1.000 chips working concurrently to achieve 1 Tera enc./s

- alternatively: compute for known plaintext all possible encryptions and use this to determine the key
  ⇒ much too high memory requirements

- alternatively: compute the possible encryptions only partially
  ⇒ trade-off between space and time

look at DES-cracker at Electronic Frontier Foundation - http://www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/

- **differential cryptanalysis:**
  use suitably chosen different plaintexts and analyse differences in the ciphertexts to obtain information on the key.
  ⇒ interesting method (published 1990 by Biham&Shamir), DES has been especially designed to resist this type of attack (this is achieved by proper design of the S-boxes);
  (but, if reduced to 8 rounds, DES is broken within minutes using differential cryptanalysis)

⇒ *it looks like the NSA has known differential cryptanalysis already in 1977!*

physical cryptanalysis:

- If DES is on a smartcard, change the (execution of the) code (or the key) by external interference. This can be done in a very specific way!
  Analyse the ciphertext which is produced by the modified execution.
  ⇒ on the average 5 key bits are retrieved per modified encryption, i.e. few attacks (<12) suffice to retrieve the key!

trivial cryptanalysis:

- Obtain the key bits directly from memory (particularly possible for DES on an (old fashioned) smartcard, if the location of the key in memory is known):
  - Simply modify the individual key bits by external manipulation and determine the correct value by testing for parity errors (very often keys are stored in 8 blocks of 7 bits using a parity bit).

Consequence:

- Software based cryptanalysis of DES requires immense effort (in 1998 "public attacks" used a hundred thousand computers on the Internet to meet the "DES challenge III" within 22 h).
- Physical attacks on smartcards can succeed quite fast.
- **Therefore, DES should be used with care** (e.g. use extensions like **triple DES**).

Main problems with DES and all symmetric methods:

- Secure communication between n persons requires $O(n^2)$ keys.
- The transmission of keys has to be done with highest security.

Alternative approach:

"Public-Key" cryptosystems

- Every person has her own public key P to be used for sending her encrypted messages (listed in a public "key book").
- Every person has her own secret key S for decrypting messages.
- Let M be a message
  C=P(M) message encrypted using P
  M'=S(C) message decrypted using S

Requirements:

(i) S(P(M))=M for every message M.
(ii) All the pairs (S,P) are different.
(iii) To derive S from P is as hard as decrypting P(M) without knowing S.
(iv) Both S and P are easily computed.
(v) Both encryption and decryption are not too expensive.

RSA-cryptosystem (Rivest, Shamir, Adleman 1978)

- public key P: pair of integers (N,c) *(c for "code")*
- secret key S: pair of integers (N,d) *(only d is secret)* *(d for "decode")*
- c and d have at most as many bits as N
- standard lengths of N: 512, 1024, or 2048 bits

RSA encryption:

1) Divide message M into sequence $m_1 \ldots m_k$ of numbers (bit blocks) smaller than N.

2) Compute $c_i := P(m_i) = m_i^c \bmod N$.

RSA decryption:

1) Divide message C into sequence of numbers $c_1 \ldots c_k$ smaller than N.

2) Compute $m_i := S(c_i) = c_i^d \bmod N$.

Choice of N, c und d: *(for N having 1024 bits)*

1) Randomly generate two large prime numbers p, q both having 512 bits.
2) Compute N:= p·q.
3) Determine (randomly) c such that c is prime relative to (p-1)(q-1).
4) Determine d such that $cd \equiv 1 \pmod{(p-1)(q-1)}$

Lemma: For these N, c, and d we have for all m < N
$m^{cd} \equiv m \pmod N$

Validity of the Lemma is due to results of number theory:

Fermat's Little Theorem ("Kleiner Satz von Fermat"):
p prime, a not divisible by p $\Rightarrow$ $a^{p-1} \equiv 1 \pmod p$

Generalisation by Euler:

For $n \in \mathbb{N}$ define $\Phi(n) = |\{m \in [n] \mid m \text{ prime relative to } n\}|$
Φ is called *Euler-function*.

Euler's Theorem:
$n,a \in \mathbb{N}$, a prime relative to n $\Rightarrow$ $a^{\Phi(n)} \equiv 1 \pmod n$

Corollary :
For all a, $n \in \mathbb{N}$ $a^{\Phi(n)-1}$ is the modular inverse of a wrt n.

Back to the RSA-algorithm:

p, q prime $\Rightarrow$ $\Phi(pq) = (p-1)(q-1)$ (explanation)

thus: m prime relative to N=pq $\Rightarrow$ $m^{(p-1)(q-1)} \equiv 1 \pmod N$

$\Rightarrow$ $m^{(p-1)(q-1)+1} \equiv m \pmod N$

$cd \equiv 1 \pmod{(p-1)(q-1)}$ $\Rightarrow$ $cd = k\cdot(p-1)(q-1)+1$

$\Rightarrow$ $m^{cd} \equiv m \pmod N$

this holds even, if m is not prime relative to N = pq :

because:
$t < p \Rightarrow t^{p-1} \equiv 1 \pmod p$ and $q^{p-1} \equiv 1 \pmod p$ (because of „little Fermat")

$\Rightarrow \forall\, k>0$ $t^{k(p-1)(q-1)} \equiv 1 \pmod p$ and $q^{k(p-1)(q-1)} \equiv 1 \pmod p$

$\Rightarrow$ $t(tq)^{k(p-1)(q-1)} \equiv t \pmod p$

$\Rightarrow$ $(tq)^{k(p-1)(q-1)+1} \equiv tq \pmod{pq}$

(analogously for all multiples of p)

next

Explanation for p, q prime ⇒ Φ(pq) =(p-1)(q-1) :

If p and q are prime, then the following numbers have a common divisor with p·q:

p·1, p·2, p·3, …, p·q, 1·q , 2·q , …, (p-1)·q

these are p+q-1 numbers.

⇒ p·q - (p+q-1)= (p-1)(q-1) is the number of integers which are prime relative to p·q

(back)

Costs of the RSA-algorithm

Encryption/decryption: use **fast exponentiation**:

For all x,n we can compute x^n with O(log n) arithmetic operations.

(method: continued squaring of x and multiplication of the computed powers according to the binary representation of n) details

but: arithmetic operations required on long integers (usually 512 to 1024 bits or even more!)

- naive multiplication of k-word integers needs $O(k^2)$ single word operations.
- simple recursive approaches lead to $O(k^{1.5})$ operations
- application of FFT leads to O(k log k) operations

⇒ for 32-bit arithmetic and exponent length of 1024 bit we need about $2 \cdot 1024 \cdot 32 \cdot 5 = 327680$ operations per plaintext block of 1024 Bit
(be aware that this is a naive but nevertheless informative estimate)

good RSA-chips and RSA-algorithms on special purpose processors (e.g. Systola 1024) achieve RSA-encryption at about 100Mbps.

next

Details of fast exponentiation:

- x arbitrary integer
- $n = \sum_{i=0}^{k} a_i \cdot 2^i$

$$x^n = x^{\sum a_i 2^i} = \prod_{i=0}^{k} x^{a_i 2^i}$$

$$= x^{a_0} \cdot \left(x^2\right)^{a_1} \cdot \left(x^4\right)^{a_2} \cdot \ldots \left(x^{2^k}\right)^{a_k}$$

$$= x^{a_0} \cdot \left(x^{a_1} \cdot \left(x^{a_2} \cdot \left(x^{a_3} \cdot \ldots \left(x^{a_k}\right)^2 \ldots \right)^2 \right)^2 \right)^2$$

These two different representations of x^n lead to two different algorithms for fast exponentiation:

- least significant to most significant bit of n
- most significant to least significant bit of n

back

Costs of cryptanalysis

Finding p and q from N and c requires **factorisation of N**, a very complex operation:

The currently best known method needs $O\left(e^{\sqrt{\ln N \cdot \ln(\ln N)}}\right)$ operations.

⇒ For sufficiently large N it is "practically infeasible" to compute p and q.

e.g. assume 10^{12} operations per sec and N with 300 decimal digits,

then - using a naive estimate based on the above formula - factorisation algorithms need a worst case time of several thousand years!

(at least with respect to current knowledge)

nevertheless: advances in algorithms and in computing may lead to drastic reductions of the time needed for factorisation:

- probabilistic methods
- quantum computing

therefore: to achieve a high level of security over a long period of time, we need very long keys (much longer than for symmetric methods)

How to determine large prime numbers?

Task:
Find the largest prime number less than or equal to n.

Sieve of Erasthostenes :

- Write down all numbers from 2 to n.
- *Repeat*
  - delete first number and all its multiples from the list

until the list consists of a single element.

⇨ unfeasible for numbers with 512 bits and more

Better:
- Choose random odd number x with 512 Bit.
- *While* (**x is not prime**)
  Increase x by 2.

But: How can we test whether x is prime?

How can we test whether x is prime?

Probabilistic algorithm by Rabin & Miller:

1. Check, whether x is divisible by some small prime.
   (e.g. for all primes less than 10.000)
2. *Repeat* at most r times
   Choose random a < x.
   until Witness(a,x)

For centuries, the complexity of primality testing has been an open problem.
In August 2002, the first polynomial time algorithm has been presented!
(by M.Agrawal and two of his students (Bachelor level!) at the IIT Kanpur)

Here, Witness (a,x) is the following test:
Let $a_k\, a_{k-1} \ldots a_0$ be the binary representation of x-1.

```
d:= 1;
FOR i:= k DOWNTO 0 DO
    t:= d;
    d:= (t*t) MOD x;
    IF d=1 AND t≠1 AND t≠x-1
        THEN RETURN TRUE;
    IF a_i=1
        THEN d:= (d*a) MOD x;
IF d≠1
   THEN RETURN TRUE
   ELSE RETURN FALSE.
```

Theorem:
x prime and $t^2 \equiv 1 \pmod x$
⇒ t=1 or t= -1

Theorem (Fermat):
x prime ⇒ $a^{x-1} \equiv 1 \pmod x$

Probability of a false result of Witness(a,x) is smaller than 1/4.

⇒ probability of a false result of the Rabin-Miller Test is $< 2^{-2r}$

⇒ suitable fast and sufficiently secure method
for generating prime numbers

How do you find numbers

c with c prime wrt (p-1)(q-1)

and d with $cd \equiv 1 \pmod{(p-1)(q-1)}$?

Answer:
1) Choose arbitrary c < (p-1)(q-1). *(not too large)*
2) Use the extended Euclidean Algorithm to test, whether gcd(c, (p-1)(q-1))=1.
This algorithm produces at the same time d and an answer to the test.

Extended Euclidean Algorithm:

task: For integers c and e compute integers x, y, and r such that cx+ey=gcd(c,e)=r.
method:
define

$a_0=c$	$a_1=e$	$x_0=1$	$x_1=0$	$y_0=0$	$y_1=1$
$q_i=\lfloor a_{i-1}/a_i \rfloor$	$a_i= a_{i-2}- a_{i-1}q_{i-1}$	$x_i= x_{i-2}- x_{i-1}q_{i-1}$		$y_i= y_{i-2}- y_{i-1}q_{i-1}$	

- let k be the smallest index s.t. $a_k=0$. Then we have $\gcd(c,e)=a_{k-1}$
- Furthermore, we have for all $i \geq 0$ $c\,x_i + e\,y_i = a_i$ (proved by simple induction)

Consequence

If gcd(c,e) = 1 , then x is the multiplicative inverse of c modulo e: $cx \equiv 1 \pmod e$

Applications of the RSA-Algorithm:

Encryption of messages:

advantages:

- Asymmetry avoids the problem of secure key exchange
- public key system allows everybody to send encrypted messages without prior communication with the receiver,
- secure communication between k partners requires only k key pairs *(compared to k(k-1)/2 for symmetric methods)*

disadvantages:

- high computational cost for encryption and decryption

Secure key exchange for symmetric methods:

Key Exchange without making use of public key methods

- Alice and Bob agree upon a common general key G
  - by phone
  - by letter
  - by courier
  - transmitted completely or in segments $G_1,\ldots, G_k$ such that $G=f(G_1,\ldots, G_k)$ *(for some function f : concatenation, addition,.....)*
- Alice and Bob use a new (random) session key S for every exchange of messages.
  - Message M is encrypted using S
  - S is encrypted using G and sent together with the message

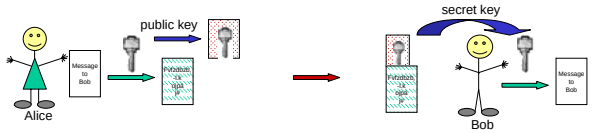
⇒ – G is used for short messages only (only for session keys)

- S is used for one message only

⇒ improved protection against cryptanalytic attacks

Using public key methods for key exchange:

- Alice and Bob
  - choose a new session key for every exchange of messages,
  - encrypt their messages using some sufficiently secure symmetric method,
  - encrypt the session key with the public key of the receiver using some asymmetric method (e.g. RSA) and send the encrypted message together with the encrypted key *(this corresponds to putting the key into a sealed envelope)*
  - only the receiver's secret key can be used to retrieve the session key which is needed to decrypt and read the message.



⇒ • there is no need for a secretly transmitted general key

- the high cost of RSA is tolerable for short session keys.
- every key of the symmetric method is used only once

Meanwhile, this (almost) is one of the most often used methods for secure communication.

Avoiding passive attacks in the Internet:

- Encrypt every datagram sent over the Internet.
- Problem:
  - Router needs information from the IP-Header

  ⇒ every router has to decrypt and encrypt the IP-Header

  ⇒ needs fast method, e.g. a fast symmetric algorithm with new session key for every datagram.

  otherwise:

  – if a key is being used on a link between two routers for a longer period of time, there are plenty chances for plaintext attacks (by sending your own datagrams which have to be sent over this link)

⇒ All this causes significant overhead leading to heavily reduced throughput.

Another symmetric method: IDEA

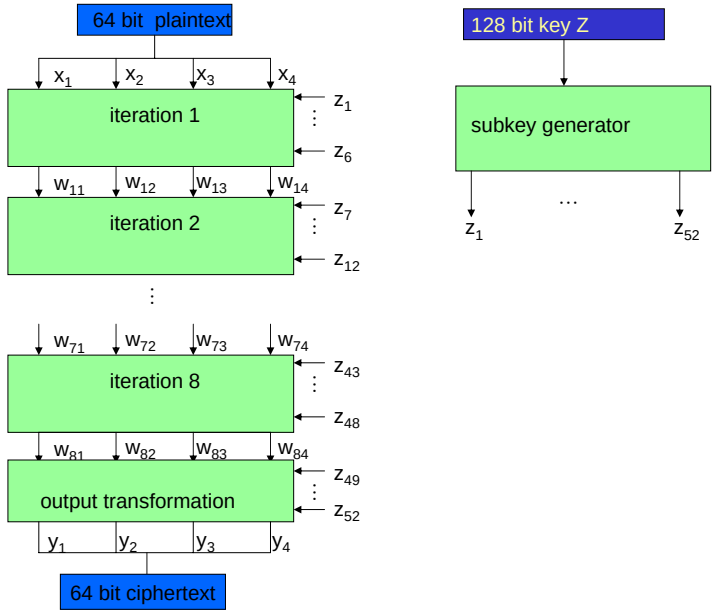
International Data Encryption Algorithm (IDEA)

- designed by Lai and Massey at the ETH 1990 - 1992
- uses key length of 128 bits
- uses data block size of 64 bits
- uses bitwise XOR ($\oplus$) and the following operations:
  - addition modulo 2^{16} (+)
  - multiplication modulo $2^{16}+1$ (•)

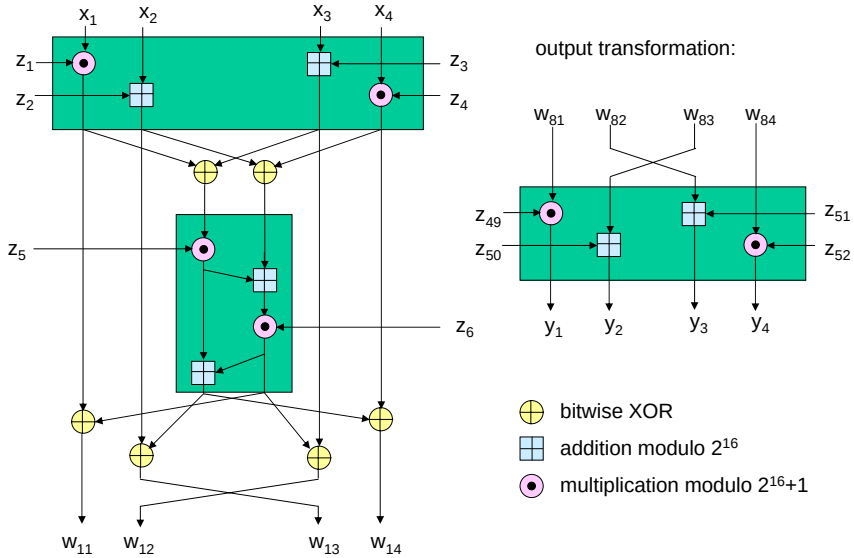
$\Rightarrow$ significantly larger diffusion effect than in DES, i.e. the individual bits of the key and of the plain text have larger influence on more bits of the ciphertext.

IDEA is used by PGP (at least in some variants of it, see further down) and therefore has found widespread application.

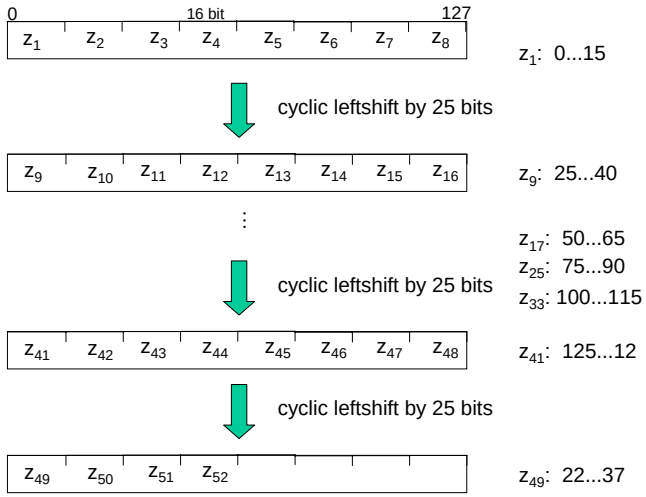
Structure of IDEA:



Structure of the iteration blocks of IDEA (the first one):



generation of subkeys:



IDEA decryption :

- use the same structure as for encryption
- use different subkeys:
  - let $z_{i1},\ldots,z_{i6}$ be the subkeys of encryption iterations i=1,…,9
    and $z'_{i1},\ldots,z'_{i6}$ the subkeys of decryption iterations i=1,…,9
    (for i=9 these are only the 4 subkeys of the output transformation)
    the decryption subkeys are chosen as follows :
    - $z'_{i1} \cdot z_{(10-i)1} \equiv 1 \pmod{2^{16}+1}$ and $z'_{i4} \cdot z_{(10-i)4} \equiv 1 \pmod{2^{16}+1}$ (i=1,…,9)
    - $(z'_{i2} + z_{(10-i)3}) \equiv 0 \pmod{2^{16}}$ and $(z'_{i3} + z_{(10-i)2}) \equiv 0 \pmod{2^{16}}$ (i=2,…,8)
    - $(z'_{i2} + z_{(10-i)2}) \equiv 0 \pmod{2^{16}}$ and $(z'_{i3} + z_{(10-i)3}) \equiv 0 \pmod{2^{16}}$ (i=1, 9)
    - $z'_{i5} = z_{(9-i)5}$ and $z'_{i6} = z_{(9-i)6}$ (i=1,…,8)

question: Are all the $z'_{i1},\ldots,z'_{i6}$ uniquely defined by the above equations?

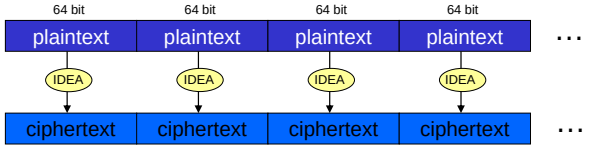
claim: The above choice of $z'_{i1},\ldots,z'_{i6}$ guarantees that the application of IDEA to the ciphertext with these subkeys produces again the plaintext.

(exercise: show this for the first iteration of decryption.)

Modes of operation (for block ciphers):

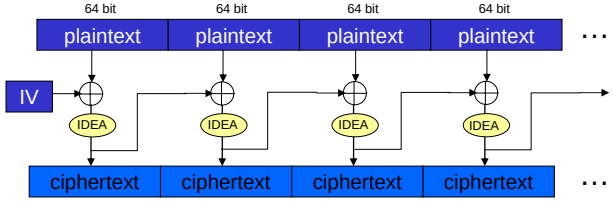
All encryption methods using block ciphers can be used in different modes of operation, in the following shown for IDEA:

- Electronic Codebook Mode (**ECB**):



- Cipher Block Chaining Mode (**CBC**):

Let **IV** be an *initial vector* (randomly chosen 64 bit).

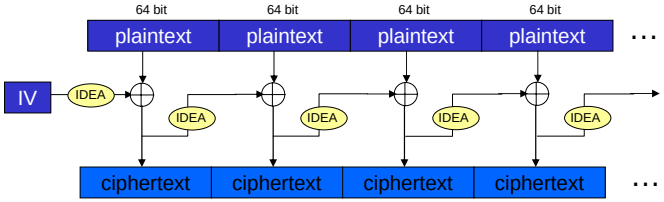


The initial vector IV is also transmitted

Question: *What kind of structure is needed for decryption?*

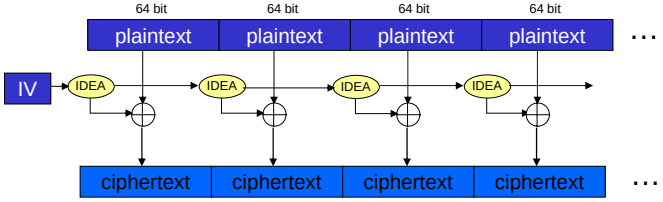
- Cipher Feedback Mode (**CFB**):

Again, let **IV** be an *initial vector* (randomly chosen 64 bits).



- Output Feedback Mode (**OFB**):

Again, let **IV** be an *initial vector* (randomly chosen 64 bits).



Advantage: the necessary encryptions of the initial vector can be computed prior to the encryption of the plaintext!

Questions, remarks on operation modi:

- For every mode of operation, what is the adequate structure for decryption?
- What is the appropriate mode of operation, if
  - you want to parallelise encryption / decryption?
  - the plaintext comes sequentially and the ciphertext has to be transmitted on-line?
  - Only parts of the ciphertext have to be decrypted (e.g. in file archives)?
- In OFB mode there is the following danger in case of repeated usage of the same initial vector:
  - The attacker got the ciphertext C belonging to the plaintext K. He manages to have the sender encrypt and send K again, but at some position at least one bit gets inserted into K.
  - Claim: Comparison of the two ciphertexts allows to retrieve the plaintext starting from the position where the first bit got inserted.
    proof: *exercise*

Further symmetric methods:

- **RC4**: is being used by some commercial products
  e.g. by Netscape for secure communication (credit card information etc.)
  - key length in the USA 128 bits, outside 40 bits (due to export regulations)
  - stream cipher
  - sensitive against attack by insertion (due to properties of XORing):

```
plaintext:      01110011010111010001110101110111111011....
random seq.:    10110101100010010100001101101110101010 1....
ciphertext:     11000110110101000101111000011001011 10....
random seq.:    10110101100010010100001101101110101010 1....
plaintext:      01110011010111010001110101110111111011....
```

  Random sequence is generated using the information from the key

- **RC5**: adjustable wrt. wordsize, number of rounds and key length
- **CAST:** generalisation of DES, quite secure
- **AES**: Advanced Encryption Standard, newly selected standard method
- ...

Advanced Encryption Standard

- Competition for determining the successor of DES
- 5 finalists in 1999
- Algorithm Name — Submitter Name(s)
  - MARS — IBM (represented by Nevenko Zunic)
  - RC6TM — RSA Laboratories (represented by Burt Kaliski)
  - Rijndael — Joan Daemen, Vincent Rijmen
  - Serpent — Ross Anderson, Eli Biham, Lars Knudsen
  - Twofish — Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson
- October 2000: announcement of winner **Rijndael**
- named after the designers Vincent **Rij**men & Joan **Dae**men (Belgium)
- Information at
  - http://csrc.nist.gov/encryption/aes/
  - Document on Rijndael available on web page of this course
- The following information has been taken from the official specification (October 2000), updated with respect to the official standard FIPS-197 (December 2001):

Parameters of Rijndael

- **Block length**: Nb * 32 : 128, 192, or 256 **(FIPS-197: 128 bits only, i.e. Nb=4)**
- **Key length**: Nk * 32 : 128, 192, or 256
- **Number of rounds** (Nr) as a function of the block and key lengths:

Nr	**Nb** = 4	**Nb** = 6	**Nb** = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

- Plaintext/Cipher-blocks (called **state**) listed along the columns of a 4xNb-array of bytes.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$

- key listed along the columns of a 4xNk-array of bytes.

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$	$k_{0,6}$	$k_{0,7}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$

General Structure of One Round

- Each round composed of four different transformations.
  In pseudo-C notation we have:

```
Round(State,RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State,RoundKey);
}
```

- The final round of the cipher is slightly different:

```
FinalRound(State,RoundKey)
{
    ByteSub(State) ;
    ShiftRow(State) ;
    AddRoundKey(State,RoundKey);
}
```

Step 1 : Byte Substitution

Based on arithmetic in the finite (Galois) field **GF(2^8):**

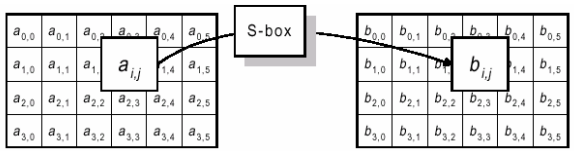
"+" = bitwise XOR-ing

"·" = polynomial multiplication modulo ($x^8 + x^4 + x^3 + x + 1$)

1. First, taking the multiplicative inverse of every byte in GF(2^8), '00' is mapped onto itself.
2. Then, applying an affine transformation over GF(2) :

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

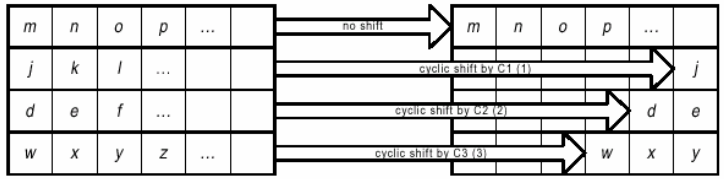
Combined step is called **S-box**:



Step 2: Cyclic Shift

- Row i of the state matrix except for row 0 is shifted cyclically by distance Ci

Nb	Shift distance for row 1: C1	Shift distance for row 2: C2	Shift distance for row 3: C3
4	1	2	3
6	1	2	3
8	1	3	4



- This is in accordance with the official specification, but be aware that in the presentation of the algorithm (at NISSC'2000) shift directions are different!

Step 3: Mix Column

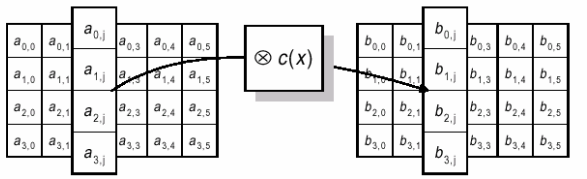
- In MixColumn, the columns of the state matrix are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial c(x), given by

  $c(x) = '03'\ x^3 + '01'\ x^2 + '01'\ x + '02'$.

  This polynomial is coprime to $x^4 + 1$ and therefore invertible.

- This can be written as a matrix multiplication:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$



Step 4: Round Key addition

- In this operation, a **Round Key** is applied to the State by a simple bitwise XOR.
- The Round Key is derived from the **Cipher Key** by means of a **key schedule**.
- The Round Key length is equal to the block length **Nb**.
- The **key schedule** is produced by **key expansion** and consists of Nb(Nr+1) words (columns of 4 bytes).
- Before Round 1 there is one initial key addition (in "Round 0"):

Cipher Key:

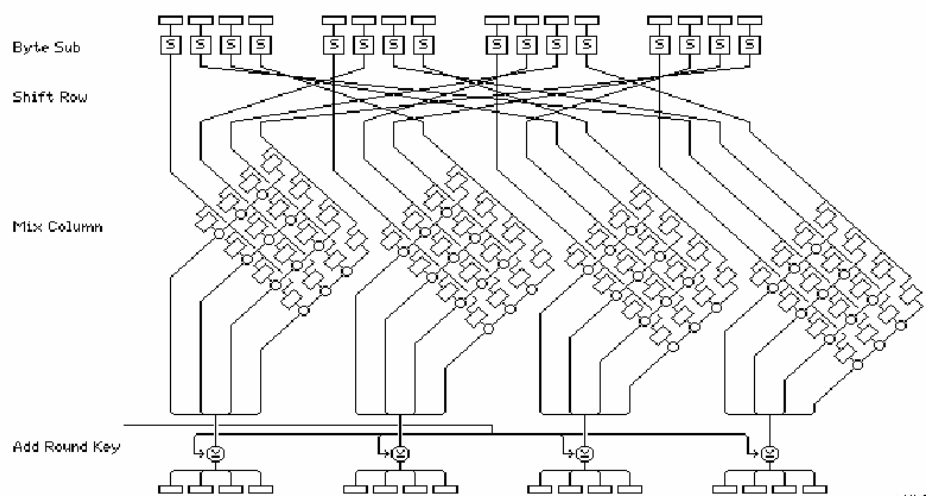
$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$	$k_{0,6}$	$k_{0,7}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$

Round Key:

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	⊕	…	$k_{0,r}$	$k_{0,r+1}$	$k_{0,r+2}$	$k_{0,r+3}$	…	=	$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$		…	$k_{1,r}$	$k_{1,r+1}$	$k_{1,r+2}$	$k_{1,r+3}$	…		$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$		…	$k_{2,r}$	$k_{2,r+1}$	$k_{2,r+2}$	$k_{2,r+3}$	…		$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$		…	$k_{3,r}$	$k_{3,r+1}$	$k_{3,r+2}$	$k_{3,r+3}$	…		$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

r= round * Nb

Rijndael in Hardware



Final remark on AES

- Rijndael has been approved as the new AES in May / June 2001
- Final specification was published November 26, 2001
- Standard became effective in May 2002
- Lots of information *(in particular tutorial and free code)* at homepage of Rijndael:
  http://www.esat.kuleuven.ac.be/~rijmen/rijndael/
  and at homepage of AES
  http://csrc.nist.gov/encryption/aes/
- Triple DES will still be around.
- Simple DES will be phased out.

Message Digests and Digital Signatures

Problem:

- Alice sends a message M to Bob.
- Bob (and any other person) would like to have a chance to check, whether the contents of M have been modified.

Example:

- Alice sends Bob an (electronic) cheque over € 5,000.
- Bob changes the amount to € 50,000 and goes to the bank.
- How can Alice make sure that the bank can check the cheque's correctness?

(Almost) safe method:

- Alice uses some function f to compute a message digest D=f(M)
- Alice sends M and D together.
- Using f, every receiver of M and D can detect whether D=f(M).
  (here, we assume for the moment that D is the original value computed by Alice.)

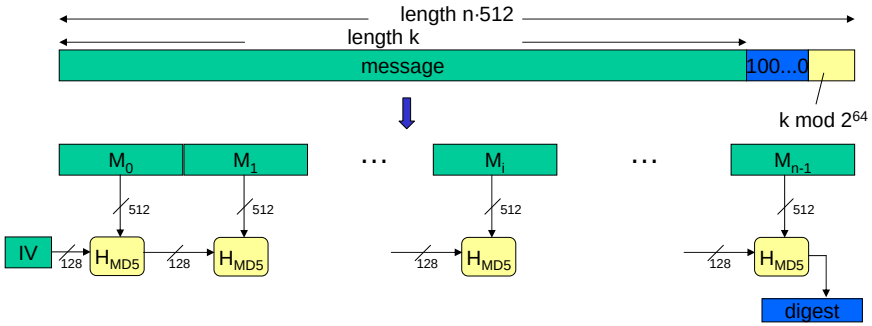
Requirements on f:

- f shall compress the contents of M significantly.
  (usually, a fixed length message digest has to be generated, e.g. 128 bits).
- For arbitrary $M \neq M'$ we should have $f(M) \neq f(M')$ with high probability
  (or, at least for all M' within some neighbourhood of M).
- Given D=f(M), it should be unfeasible to compute M.
- Given M and f(M) it should be unfeasible to compute a message M' with f(M)=f(M').

Functions like this are called **one-way hash functions**
(sometimes also compression function, concentration function, cryptographic checksum, message integrity check,...)

Well known one-way hash function:

- **MD5**: *message digest function*, described in RFC1321
  1. Extend M by a sequence from 10* to get length congruent 448 (mod 512).
  2. Append length (additional 64 bits).
  3. Compute the digest as follows:



Structure of H_{MD5}:

– sequence of arithmetic and logical operations on the four 32-bit parts of the (intermediate) digest.

SHA: Secure Hash Algorithm

– based on the same predecessor as MD5

– generates digest of length 160 Bit

– about 25% slower, but simpler structure

– part of DSA, the digital signature algorithm within DSS, the digital signature standard *(see next slide on digital signatures)*

Digital Signature:

Just the application of a good one-way hash function f is not sufficient to satisfy all the requirements:

– If Alice sends M and D=f(M), then Bob could change M into M‘, compute f(M‘), and then claim to a third party that M’ (together with f(M‘)) is the valid message.

– In the same way, Alice could pretend to Bob (or others) that she never sent M.

Remember: A “real signature”

– uniquely identifies the person and

– transforms a piece of text into a document with a certified contents (any correction is invalid!).

Algorithm for a digital signature:

Alice computes D=f(M), encrypts D using her private key (e.g. using RSA), and then sends M and her **digital signature σ** = S(f(M)) (the encrypted message digest)

Bob computes the message digest of the received message, decrypts σ using the public key of Alice, and compares the results.

⇒ **as long as the private key remains secret, neither Alice can deny to have sent M, nor can Bob pretend to have received a different message.**

Man-in-the-Middle Attack:

Application of the digital signature can only guarantee the authenticity of message and sender as long as

– the private key of Alice remains secret

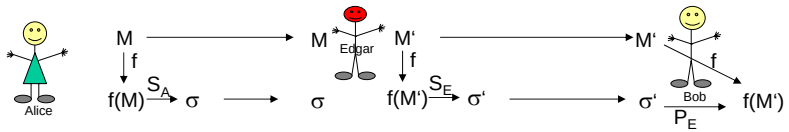
– Bob uses the public key of Alice (and vice versa)

Attacker Edgar (**man-in-the-middle**) could

– intercept and manipulate all the messages between Alice and Bob,

– send to Alice his own public key pretending it to be Bob’s,

– send to Bob his own public key pretending it to be Alice’s.



⇒ Edgar can manipulate the communication between Alice and Bob, without them noticing anything.

Precaution: Interlock Protocol *(Rivest, Shamir 1984)*

Method for secure exchange of public keys:

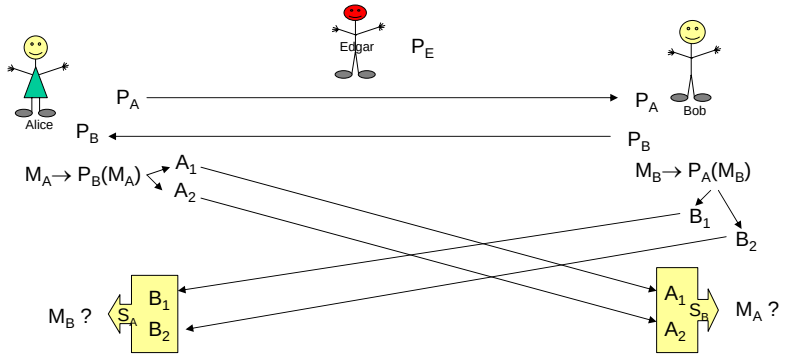
1) Alice and Bob send each other their public keys P_A and P_B.
2) Alice and Bob use these public keys to encrypt short (personal) messages (M_A and M_B).
3) Alice sends Bob some fraction of $P_B(M_A)$ which cannot be decrypted (e.g. every other byte).
4) Having received this fraction, Bob sends Alice some fraction of $P_A(M_B)$ which cannot be decrypted (e.g. every other byte).
5) Having received Bob's "submessage", Alice sends the remaining part of her message.
6) Bob reassembles both parts. If he can decrypt the message, using his private key, he can be sure that his public key has been correctly transmitted, and he sends the remaining part of his message to Alice.
7) Alice reassembles both parts. If she can decrypt the message using her private key, she can be sure that her public key has been transmitted correctly.

Argument:

The Man-in-the-Middle attack fails, since Edgar does not know M_A and M_B and since he cannot decrypt fractions of messages which have been encrypted using his public key and afterwards re-encrypt these fractions using P_B or P_A , respectively.

But: This approach relies on the secrecy of M_A and M_B !

Visualisation of the interlock protocol:



- If Edgar does not forward A_1 (or some X), Bob will not send B_1.
- If Edgar does not forward B_1 (or some Y), Alice will not send A_2.
- If Bob cannot decrypt the combined message into M_A, he will not send B_2 or any other confidential messages using P_A.
- If Alice cannot decrypt the combined message into M_B, she will not send any other confidential messages using P_B.
- Therefore, whenever Edgar tries to interfere by using his key P_E, and by delaying or modifying messages A_i or B_i, this will be detected.

Alternative to the interlock protocol: Certification

- Certification of public keys by a
  - certification authority ("Zertifizierungsstelle", CA): officially appointed institution which guarantees the correctness of public keys, i.e. the association of keys with persons. A CA issues a
    - key certificate: information on a person's public key and some additional information and a digital signature of this (using the secret key of the CA, i.e. the CA's public key must be a trusted key, or there must be ways of checking the correctness of the CA's public key)
    - attribute certificate: additional information on a person having a key certificate.

    The public key of the CA may be certified by a "next level" CA.
  - trusted person: if you know somebody's public key and get a key certificate signed using this person's secret key, then you can trust this certificate. (this is used in the "web of trust" of PGP, see below)
- Obviously, trust must be based on some verifiable information.
- Infrastructure and rules for certification of public keys are main topics of "digital signature legislation" (in Germany "Signaturgesetz", August 1997, new European regulations have been agreed upon in 1999)
- Updated legislation in Germany in the **„Deutsche Signaturgesetz"** (16. Mai 2001) and in the „**Verordnung zur elektronischen Signatur"** (available at **http://jurcom5.juris.de/bundesrecht/sigg_2001)**

Pretty Good Privacy – PGP

Developed by Phil Zimmermann (USA) independent from (large) companies or public institutions (in 1991)

Goal: Everybody should be able to send confidential messages by e-mail. Privacy protection against state supervision (CIA, FBI, secret service,...)

- original version uses RSA, IDEA, and MD5 (newer versions also use Diffie/Hellman-ElGamal, tripleDES, CAST, AES)
- runs on a large number of operating systems, usually as a plug-in to other standard software (Eudora, Explorer, Outlook,...)
- freely available (see www.pgpi.com) for private use, low fee for commercial use
- offers
  - encryption/decryption (RSA + IDEA+AES),
  - digital signature (RSA + MD5),
  - compression (ZIP),
  - Radix 64 conversion (i.e. cipher text to ASCII),
  - message segmentation,
  - generation and administration of keys for RSA,
  - generation of session keys for IDEA, CAST, AES.

How does PGP work?

- Every user has her own passphrase (maximally 253 characters).
- Using MD5 PGP generates from this a 128 Bit key.
- This key is used to encrypt the private RSA-key (using IDEA).

Generation of keys for RSA:

- *generation of "randomness"*:
  uses time differences during input of arbitrary text, characters,
  and internal time (using XOR and MD5 in CFB-mode),
  this is done once.
- *generation of prime numbers*:
  - generation of numbers using the initially "generated randomness"
  - test for divisibility by primes less than 8192 (=2^{13})
  - five random tests with "Little Fermat" (i.e. weaker than Rabin-Miller Test)
- *public key*: at least 5 bits (> 17)

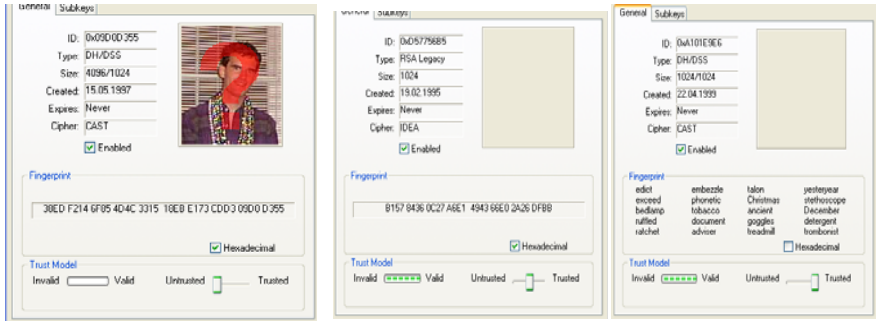
Administration of RSA keys

Certification of keys by "Web of Trust" (dynamically generated „institution")

- every PGPuser has her public key signed (certified) by other trusted persons and passes it on together with these signatures.
- If Alice adds Bob's public key, PGP asks:
  Do you acknowledge key certificates with Bob's signature?
- Alice may answer:
  1. Yes, always
  2. Sometimes
  3. No
  4. Don't know
- unknown key is assumed to be trustworthy , if it has been certified
  – by one level 1 signature
  – by two level 2 signatures

Additional test of public keys by checking their "fingerprint" (MD5).

Examples for information on public keys in the current pgp system:



Application of IDEA within PGP:

- usage of IDEA in CFB mode
- new session key and initial vector for every encryption
- session key is RSA-encrypted with the receiver's public key

Advantage of PGP:

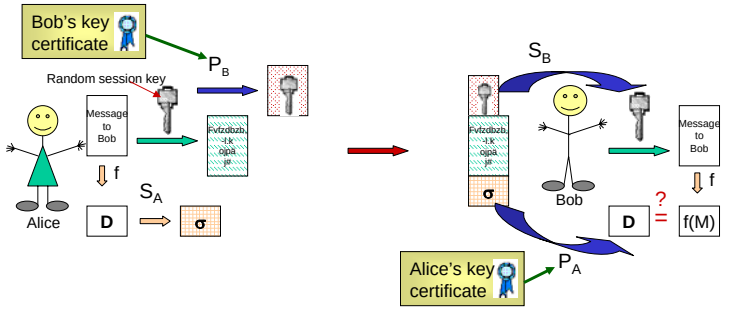
- no public (state) certification authorities for keys
  (in contrast to PEM(Privacy Enhanced Mail)), thus independent of public legislation
- freely available
- usable for encrypting and/or signing E-Mail and arbitrary files

⇒ **important tool for privacy protection and for secure management of electronic documents for private and commercial use.**

Summary:
Standard protocol for confidential communication:

1. Alice and Bob exchange their public keys using some method of certification (interlock protocol, certification authority, or Web-of-Trust, using digital signatures of the public keys (verified with a trusted public key)).
2. **Sending** a (confidential) message M (from Alice to Bob):
   - Compute the message digest D=f(M) using some suitable hash function f.
   - Generate session key k_S and encrypt M using some symmetric encryption method (IDEA, triple DES, CAST,...).
   - Encrypt k_S using Bob's (certified) public key P_B.
   - Encrypt D using Alice's private key S_A.
   - Send $S_A(D)$, $P_B(k_S)$ and $k_S(M)$.
3. **Receiving** a message:
   - Open the "electronic envelope" using Alice's (certified) public key P_A and retrieve D.
   - Decrypt $P_B(k_S)$ using Bob's private key S_B and then use k_S to decrypt M.
   - Compute f(M) and check for equality with D.

The final protocol visualised:



Assuming properly certified public keys P_A and P_B this protocol provides

- **authenticity**: requires the use of private keys S_A and S_B
- **integrity**: valid digital signature requires the use of private key S_A
- **confidentiality**: required key k_S can only be retrieved using private key S_B

For additional security wrt "birthday attacks" (modified document having the same message digest), the encrypted session key could also be signed using Alice's private key S_A (or Alice's signature could be encrypted using Bob's public key).