Multimedia Security

Authentication, digital signatures, PRNG

Mauro Barni
University of Siena

Beyond confidentiality

- Up to now, we have been concerned with protecting message content (i.e. confidentiality) by means of encryption.
- Will now consider how **to protect message integrity** (i.e. protection from modification by unauthorized parties), and how to **confirm the identity of the sender**
- Sometimes these problems are more important than confidentiality

Authentication

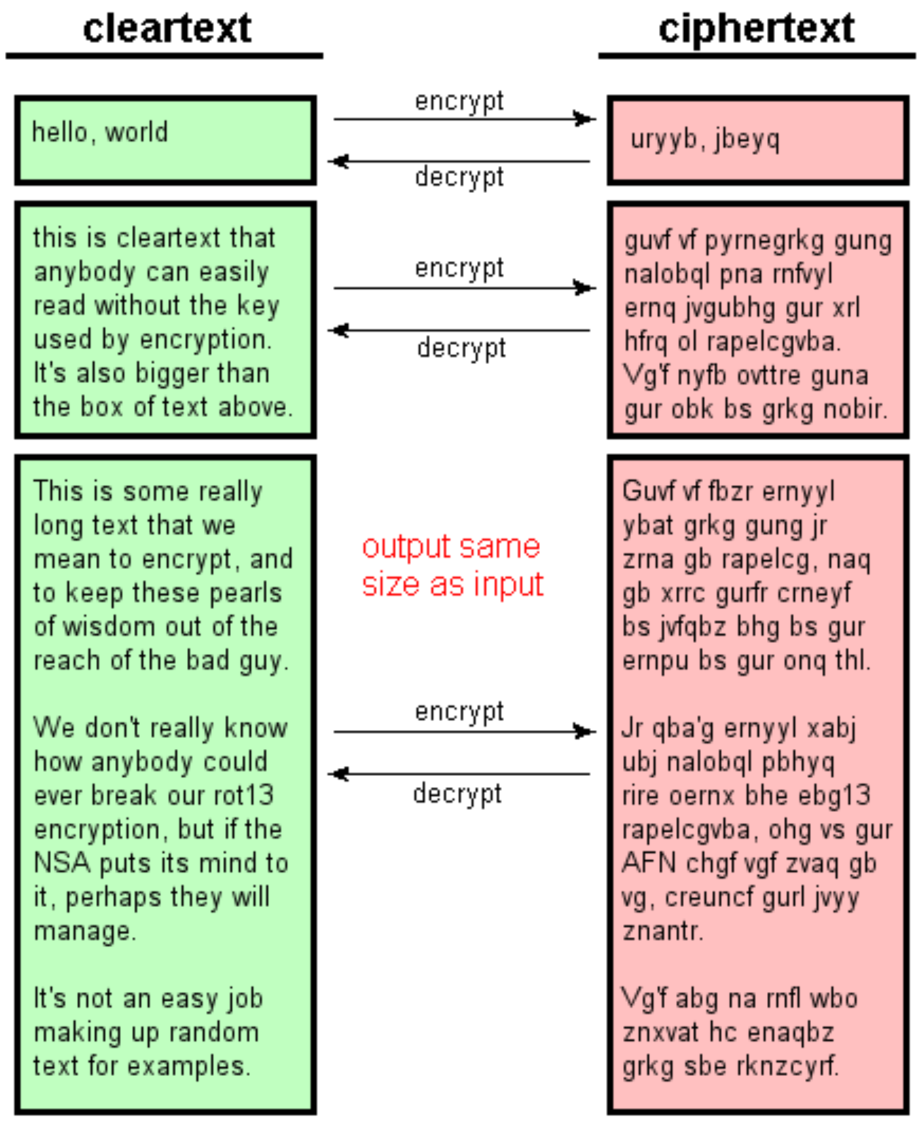
- Authentication mechanisms consist of 2 levels:
  - *low level*: a function produces an authenticator, i.e. a value to be used for message authentication;
  - *high level*: an authentication protocol uses the low level function to allow the recipient to verify the authenticity of the message.

The authenticator

- 3 alternative low level functions can be used:
  - *message encryption*: the authenticator is the encryption of the whole message;
  - *message authentication code (MAC)*: the authenticator is the output (of fixed length) of a public function having as input the message and a secret key.
  - *hash function*: the authenticator is the output (of fixed length) of a public function having as input the message.

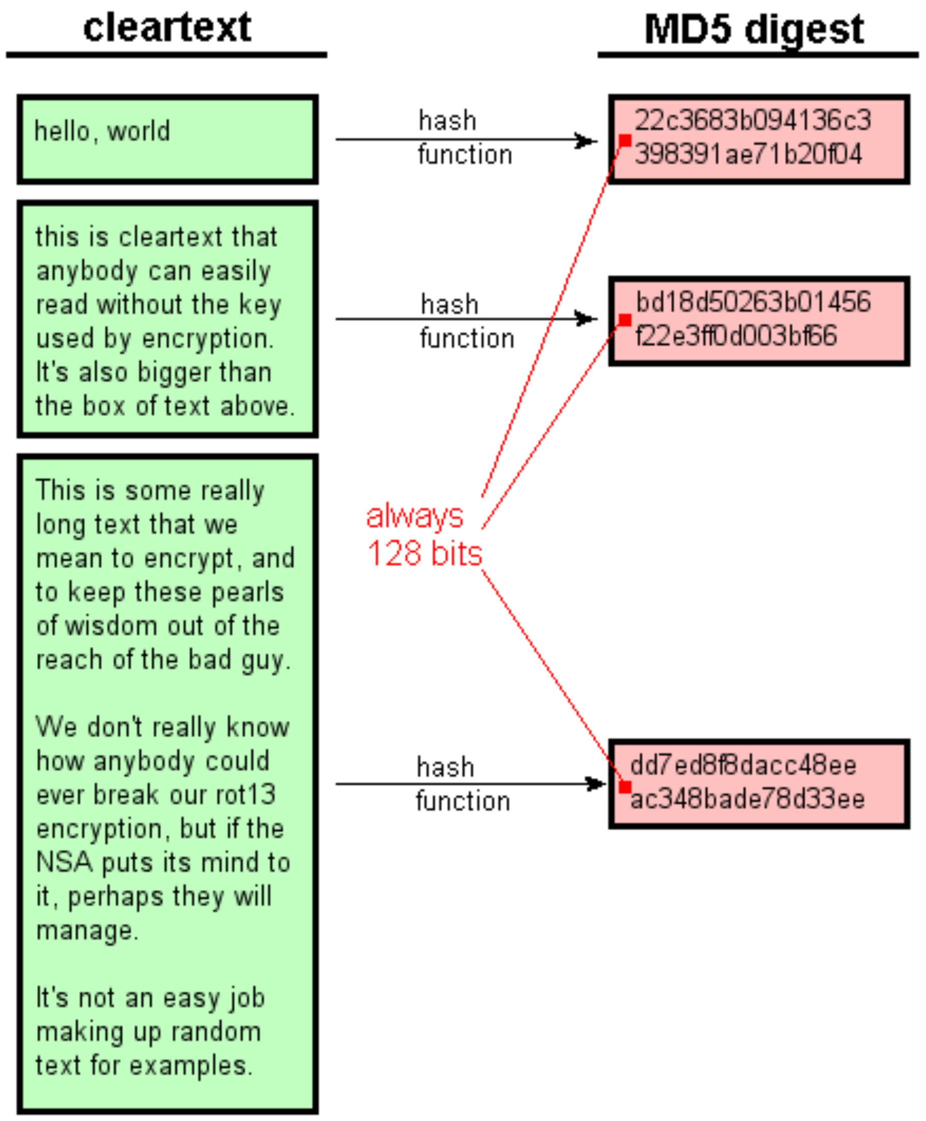
Encryption

- **Encryption** transforms data from a cleartext to ciphertext and **back**
- Big plaintext yields big ciphertext, and so on.
- "Encryption" is a **two-way** operation.


cleartext
ciphertext
hello, world
encrypt
decrypt
uryyb, jbeyq
this is cleartext that anybody can easily read without the key used by encryption. It's also bigger than the box of text above.
encrypt
decrypt
guvf vf pyrnegrkg gung nalobql pna rnfvyl ernq jvgubhg gur xrl hfrq ol rapelcgvba. Vg'f nyfb ovttre guna gur obk bs grkg nobir.
This is some really long text that we mean to encrypt, and to keep these pearls of wisdom out of the reach of the bad guy. We don't really know how anybody could ever break our rot13 encryption, but if the NSA puts its mind to it, perhaps they will manage. It's not an easy job making up random text for examples.
output same size as input
encrypt
decrypt
Guvf vf fbzr ernyyl ybat grkg gung jr zrna gb rapelcg, naq gb xrrc gurfr crneyf bs jvfqbz bhg bs gur ernpu bs gur onq thl. Jr qba'g ernyyl xabj ubj nalobql pbhyq rire oernx bhe ebg13 rapelcgvba, ohg vs gur AFN chgf vgf zvaq gb vg, creuncf gurl jvyy znantr. Vg'f abg na rnfl wbo znxvat hc enaqbz grkg sbe rknzcyrf.

Hash/MAC Functions

- Hashes/MACs, on the other hand, transform a stream of data into a small **digest** (a summarized form), and are strictly **one way operation**.
- Outputs have the same size no matter how big the inputs are

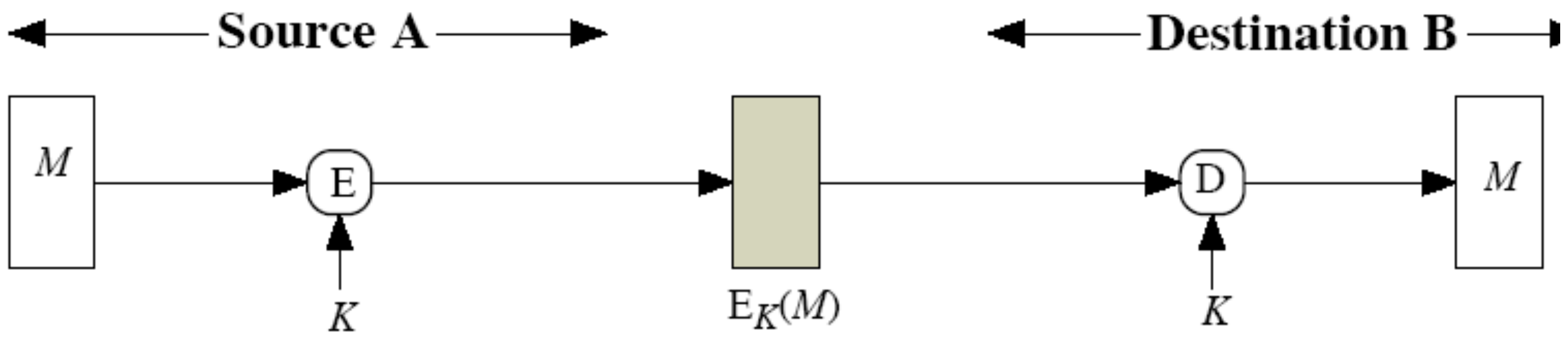

cleartext
MD5 digest
hello, world
hash function
22c3683b094136c3 398391ae71b20f04
this is cleartext that anybody can easily read without the key used by encryption. It's also bigger than the box of text above.
hash function
bd18d50263b01456 f22e3ff0d003bf66
This is some really long text that we mean to encrypt, and to keep these pearls of wisdom out of the reach of the bad guy.
We don't really know how anybody could ever break our rot13 encryption, but if the NSA puts its mind to it, perhaps they will manage.
It's not an easy job making up random text for examples.
hash function
dd7ed8f8dacc48ee ac348bade78d33ee
always 128 bits

Message Encryption

- Encryption by itself provides a way to authenticate a message
- The authenticator is the encrypted full message.
- It can be obtained through:
  - Secret-key encryption
  - Public-key encryption

Message Encryption

- If symmetric encryption is used



(a) Symmetric encryption: confidentiality and authentication

Message Encryption

– Only A and B know the key: *confidentiality*

– Receiver knows the sender must have created the message: *sender authentication*

– Receiver knows the content has not been altered: *message integrity*

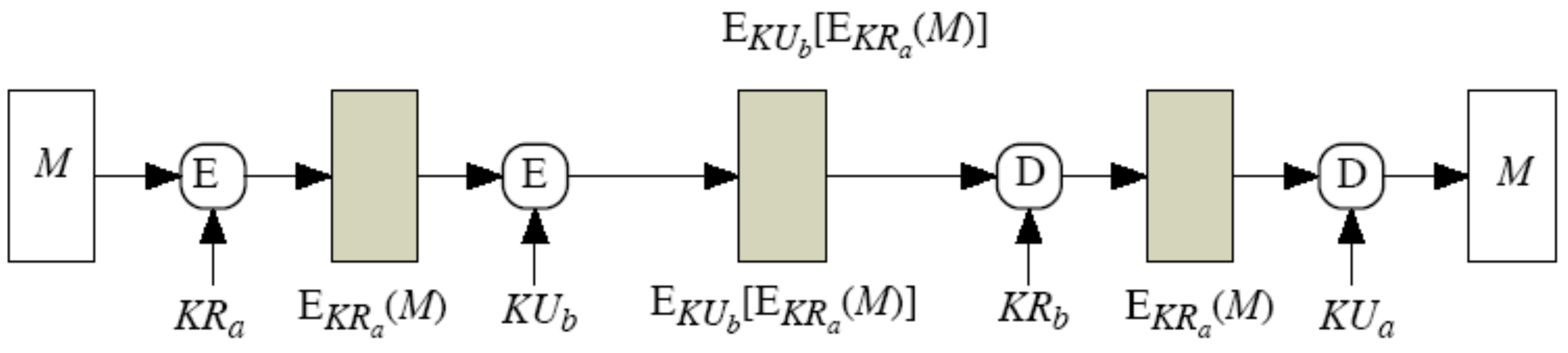
  - Any modification by an attacker without key produces evident alteration

– Drawbacks

  - It is difficult to determine in an automatic way if the decrypted message is intelligible
  - Non-repudiability is not granted

Message Encryption

- If public-key encryption is used:
  - sender encrypts message using his/her private-key
  - then encrypts with recipients public key
  - achieves both secrecy and authentication
- Two public-key encryptions used on the entire message
- Need to recognize corrupted messages
- Non-repudiability achieved



(d) Public-key encryption: confidentiality, authentication, and signature

Message Authentication Code

- A MAC is a cryptographic checksum of the message M, generated by means of a public function C:

$$\mathbf{MAC} = C_K(\mathbf{M})$$

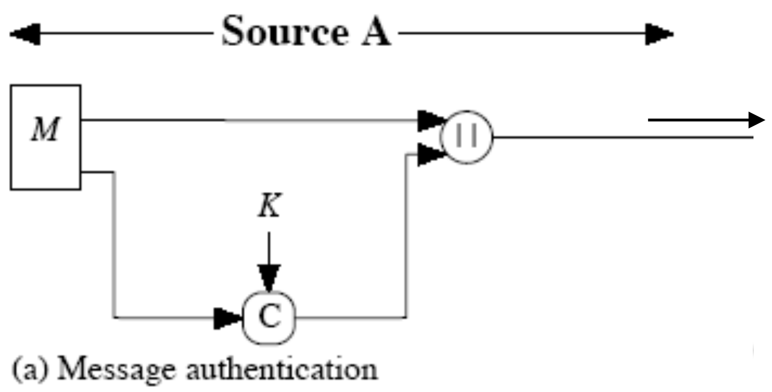
- It condenses a variable-length message M into a fixed-sized authenticator MAC (length **n** bits) by using a secret key K
- ***The output depends on input M and K, that must be shared by A and B !!!***

Message Authentication Code (MAC)

- The function C is not invertible (unlike a cryptographic function)
- Usually it is a many-to-one function: potentially many messages can have same MAC:
  - Given N messages, and MAC length = n bits, we have 2^n possible MACs, but usually $N >> 2^n$
  - Finding two messages with the same MAC must be very difficult

Message Authentication Code (MAC)

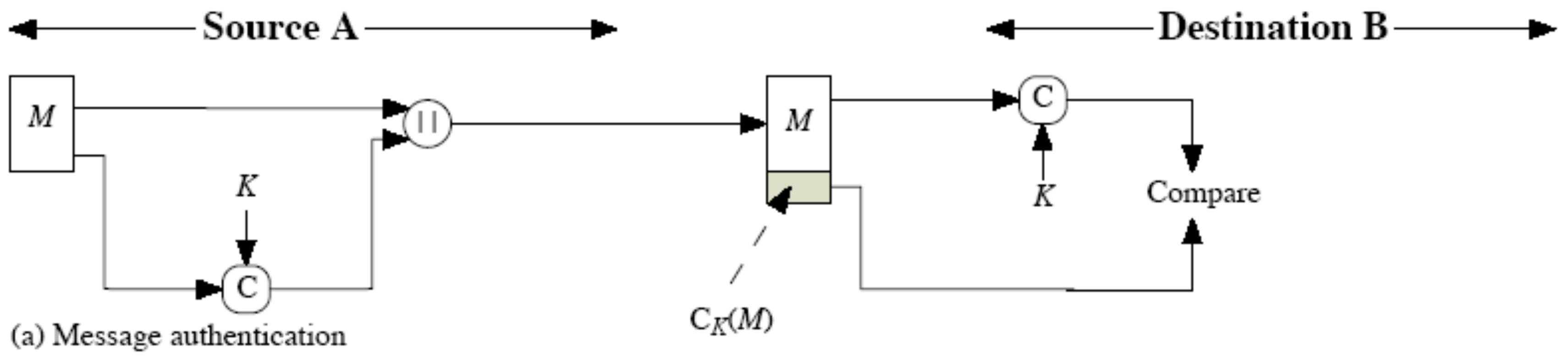
- The MAC is appended by the sender to the message before its transmission



(a) Message authentication

Message Authentication Code (MAC)

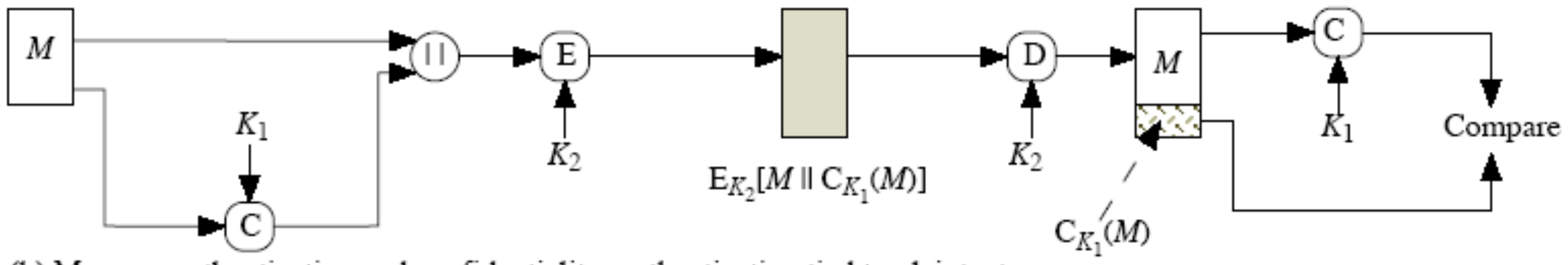
- The receiver performs the same computation on the received message and checks if it matches the received MAC:
  - It ensures message integrity (through MAC)
  - It ensures that the message comes from the sender (through K)



(a) Message authentication

Message Authentication Code (MAC)

- In the previous protocol, the MAC does not provide secrecy, since the message is transmitted unprotected.
- It is possible to add encryption to achieve secrecy
  - generally different keys for each process are used

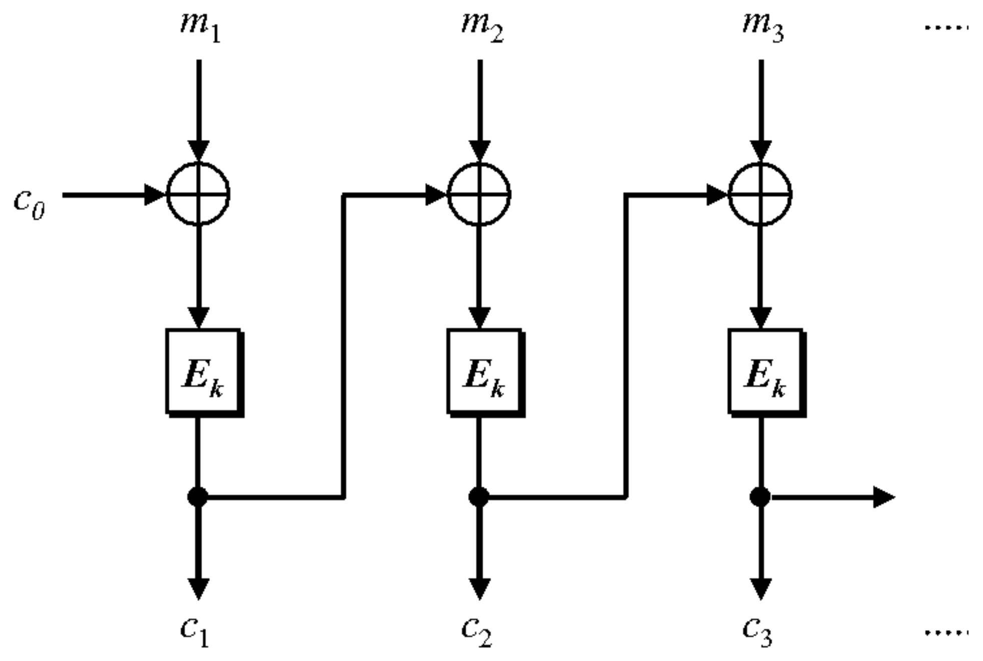


(b) Message authentication and confidentiality; authentication tied to plaintext

Symmetric Ciphers for MACs

- We can obtain a MAC with any cipher block chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** is a widely used MAC based on DES-CBC

$c_i = E_k(m_i \oplus c_{i-1})$



Message Authentication Code (MAC)

- Why using a MAC instead of symmetric crypto?
  - In some applications only authentication is needed;
  - Some applications need authentication to persist longer than encryption (eg. archival use);
  - In a flexible protocol confidentiality and authentication functionalities need to be separated.

Hash Function

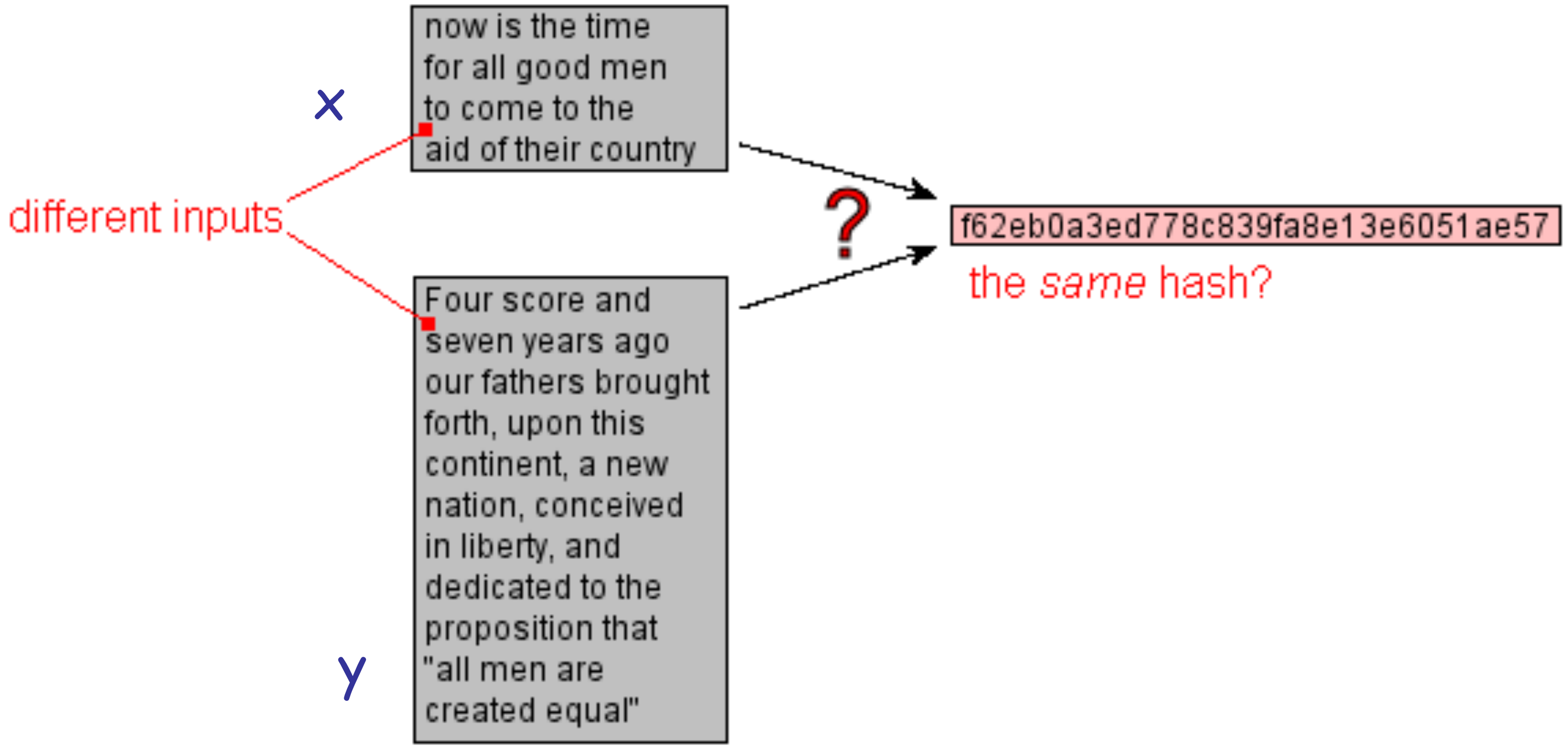
- A *hash function* is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called *hash-values* or *message digests*.
- The hash function is public and not keyed
  - output depends only on input, (cf. MAC is keyed)
- Hash is used to detect changes to message: a change in 1 or more bits in M produces a different hash.

Requirements for Hash Functions

- It can be applied to any sized message **M**
- It produces fixed-length output **h**
- **h=H(M)** is easy to compute for any message **M**
- Strong one-way property: it exhibits collision resistance

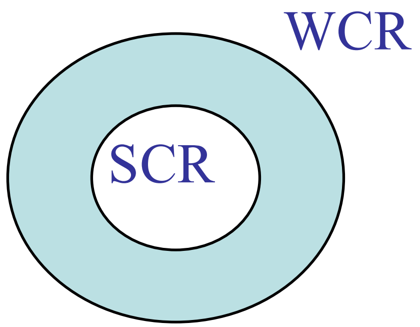
Collision resistance

- It is infeasible to find any **x,y** s.t. **H(y) = H(x)**
  - **strong collision resistance**


x
now is the time for all good men to come to the aid of their country
different inputs
?
f62eb0a3ed778c839fa8e13e6051ae57
the *same* hash?
Four score and seven years ago our fathers brought forth, upon this continent, a new nation, conceived in liberty, and dedicated to the proposition that "all men are created equal"
y

Collision resistance

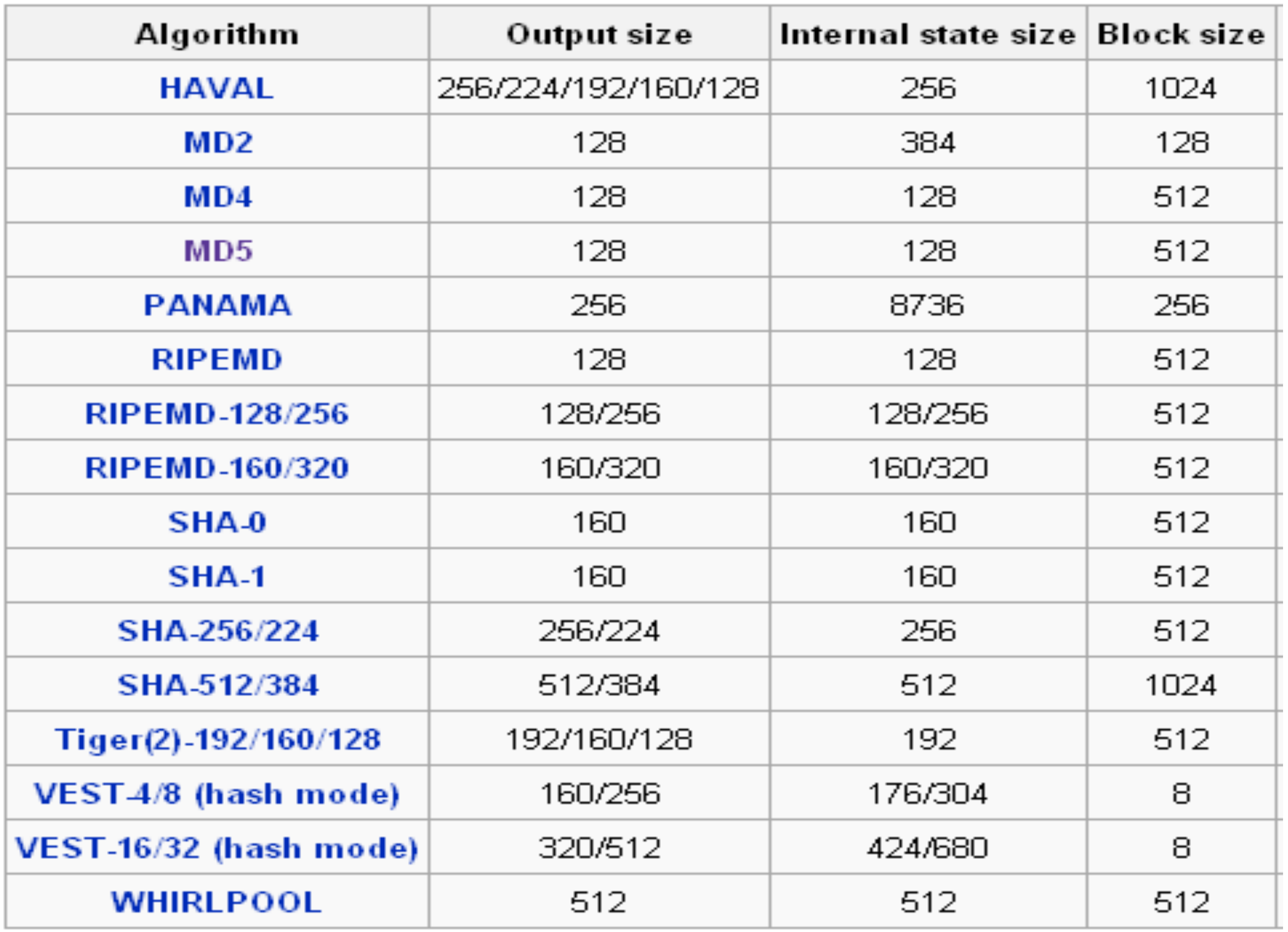
- Given **x** it is infeasible to find **y** s.t. **H(y)=H(x)**
  - weak collision resistance
- Here the adversary has a more difficult time finding collisions since it must collide on a particular input rather than any input
- It is possible to demonstrate that:
  - Strong CR => Weak CR
  - Weak CR does not imply Strong CR



Block Ciphers as Hash Functions

- We can use block ciphers as hash functions
  - using $H_0=0$ and zero-pad of final block
  - compute: $H_i = E_{M_i} [H_{i-1}]$
  - use final block as the hash value
  - similar to CBC but without a key
- A problem if we use DES is that the resulting hash is too small (64-bit)

A zoo of hash functions



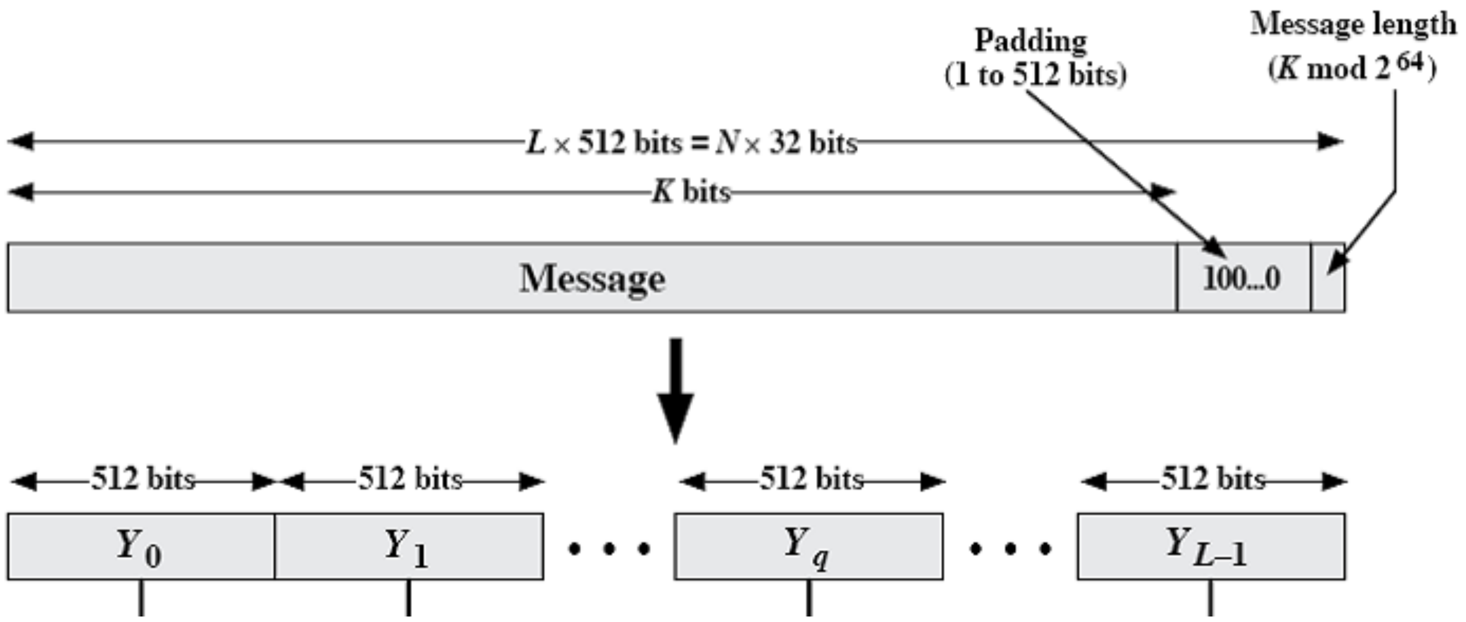
Algorithm	Output size	Internal state size	Block size
HAVAL	256/224/192/160/128	256	1024
MD2	128	384	128
MD4	128	128	512
MD5	128	128	512
PANAMA	256	8736	256
RIPEMD	128	128	512
RIPEMD-128/256	128/256	128/256	512
RIPEMD-160/320	160/320	160/320	512
SHA-0	160	160	512
SHA-1	160	160	512
SHA-256/224	256/224	256	512
SHA-512/384	512/384	512	1024
Tiger(2)-192/160/128	192/160/128	192	512
VEST-4/8 (hash mode)	160/256	176/304	8
VEST-16/32 (hash mode)	320/512	424/680	8
WHIRLPOOL	512	512	512

MD5

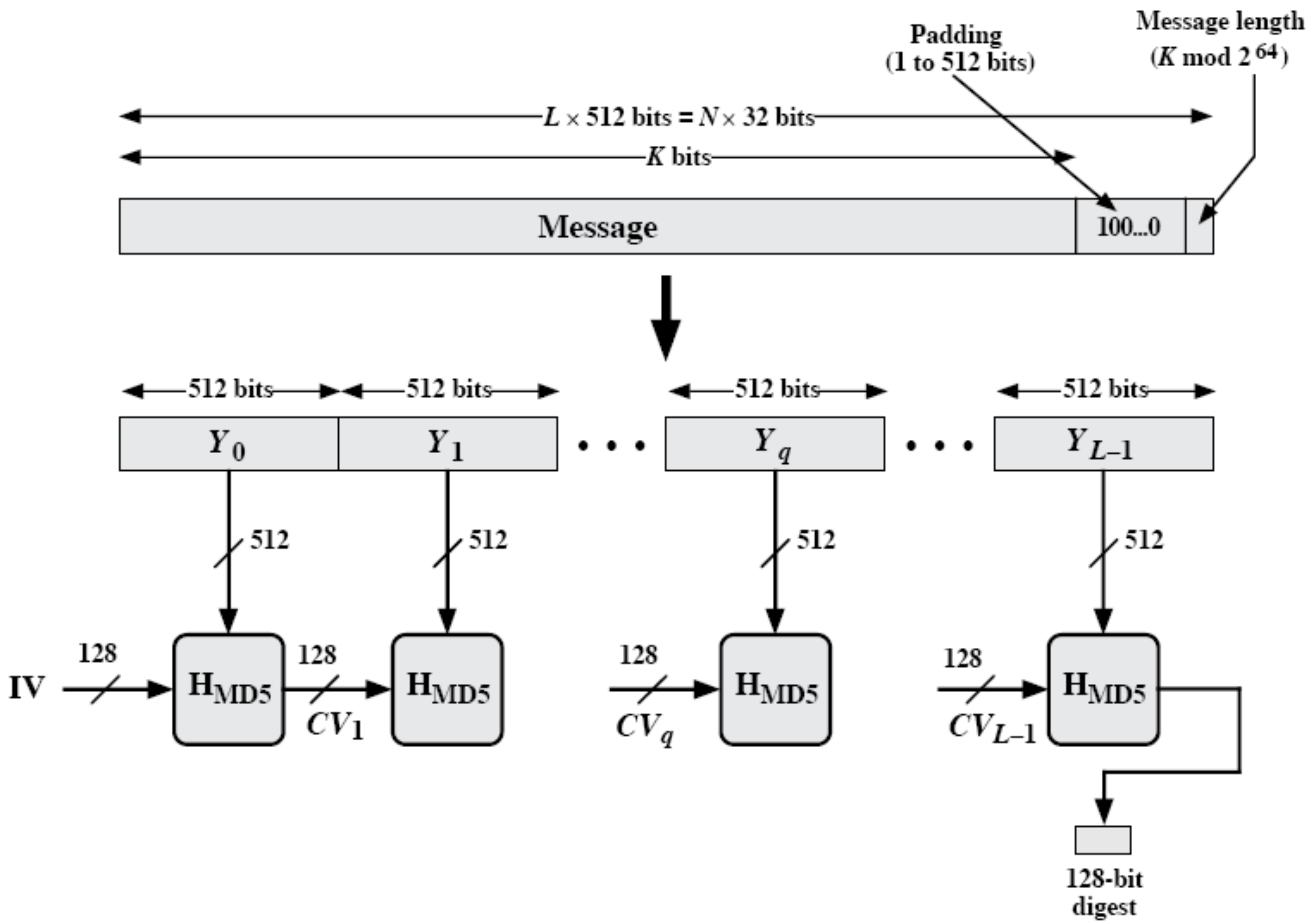
- Designed by Ronald Rivest (the R in RSA)
- Latest in a series of MD2, MD4
- It works on *input* blocks of *512* bits, and it produces a *128-bit hash value*
- Until recently was the most widely used hash algorithm in various standards and applications
  - recently security flaws were identified and SHA-2 is preferred now

MD5 Overview

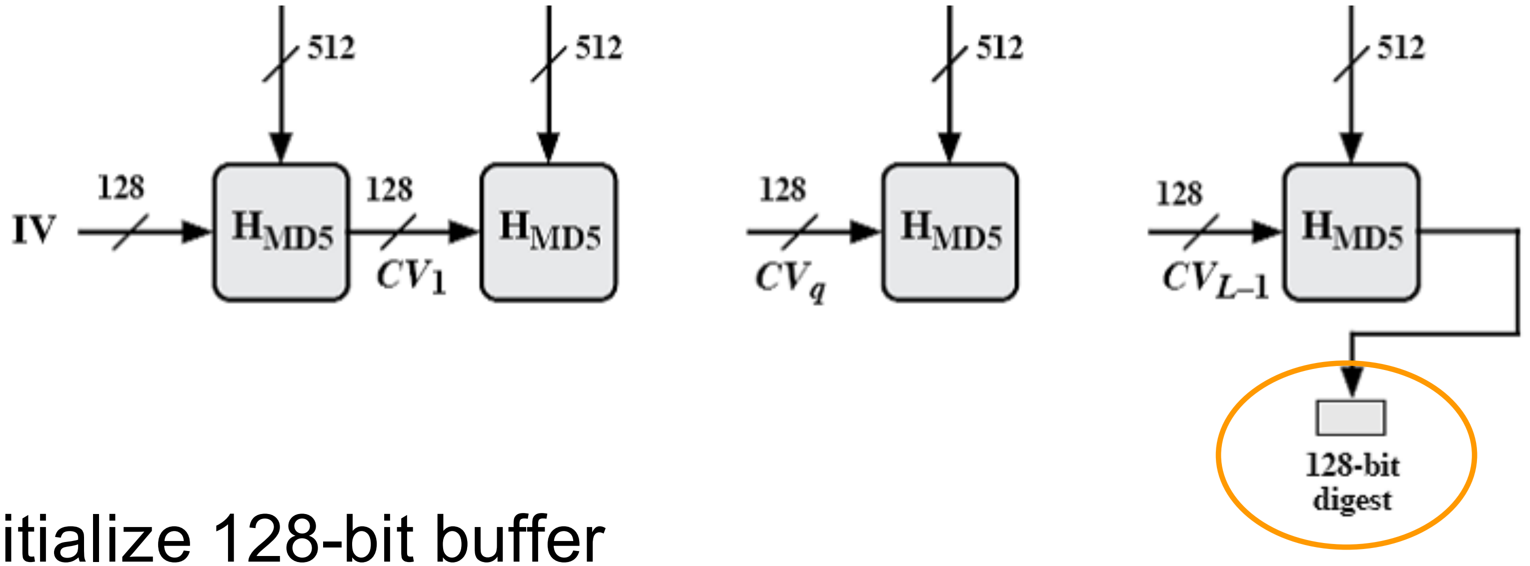
- Pad message so that its length is 448 mod 512 bits
  - if length already ok, 512 bits added (padding = 1÷512 bits)
- Append 64-bit representing original message length value (before padding, the value is mod 2^{64})
- The obtained message has a length = L x 512



MD5 Overview

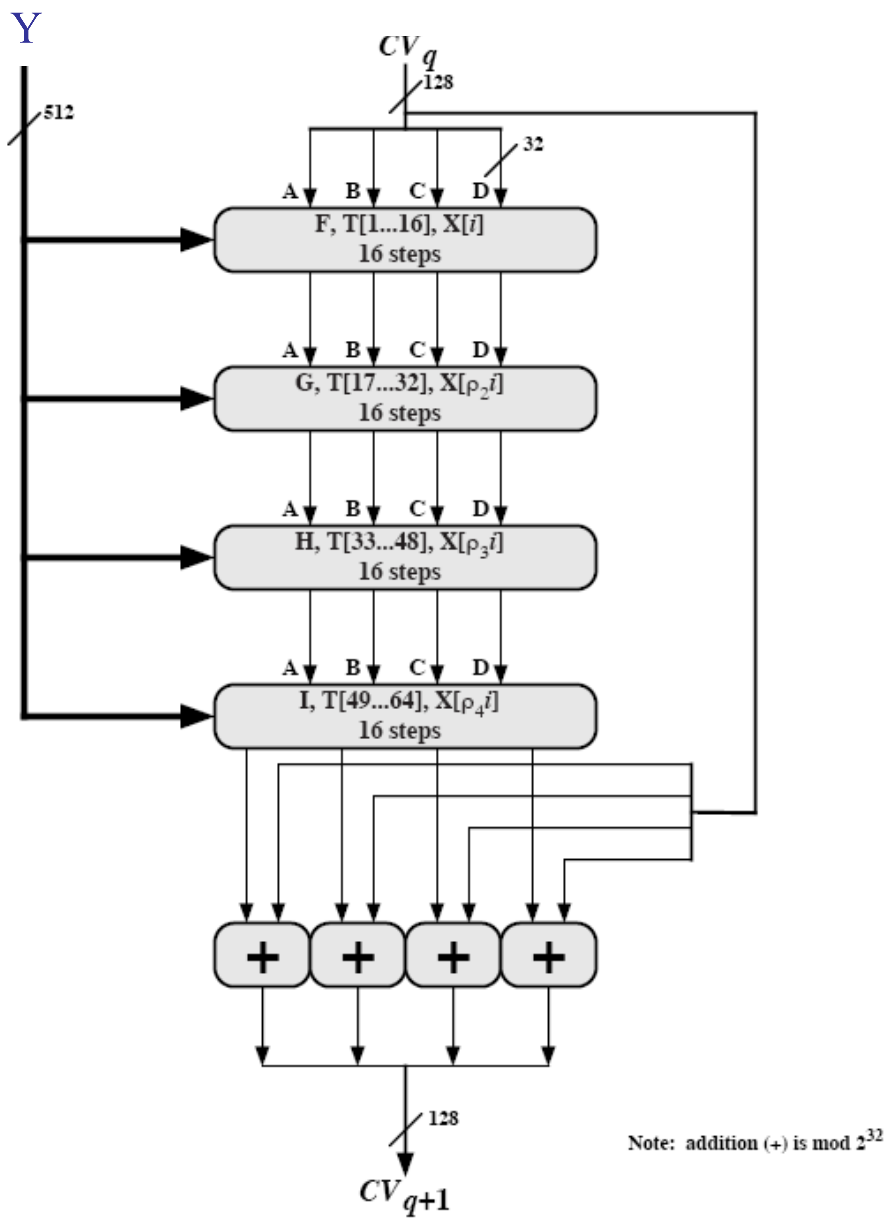


MD5 Overview



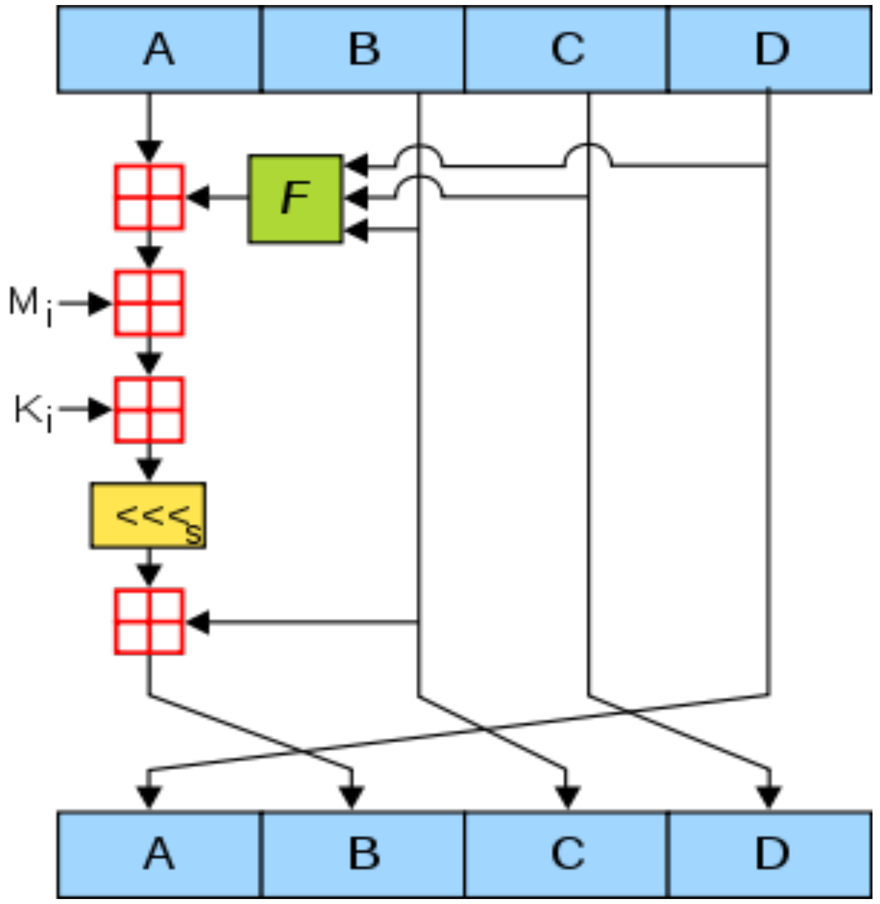
- Initialize 128-bit buffer
  - intermediate results stored in a 128 bits buffer, represented as 4 registers of 32 bits (A,B,C,D), initialised with fixed 32 bits integer numbers
- Process message in 512-bit blocks:
  - the core of the algorithm is a compression function (H_{MD5}) composed by 4 rounds on input message block & buffer;

H_{MD5} Compression Function



- Given by 4 rounds
- Each round has as input the block Y_q and the buffer value $CV_{q;}$
- The final output is added to the buffer value, to obtain the new value of the buffer CV_{q+1}
- Each round involves 16 subrounds

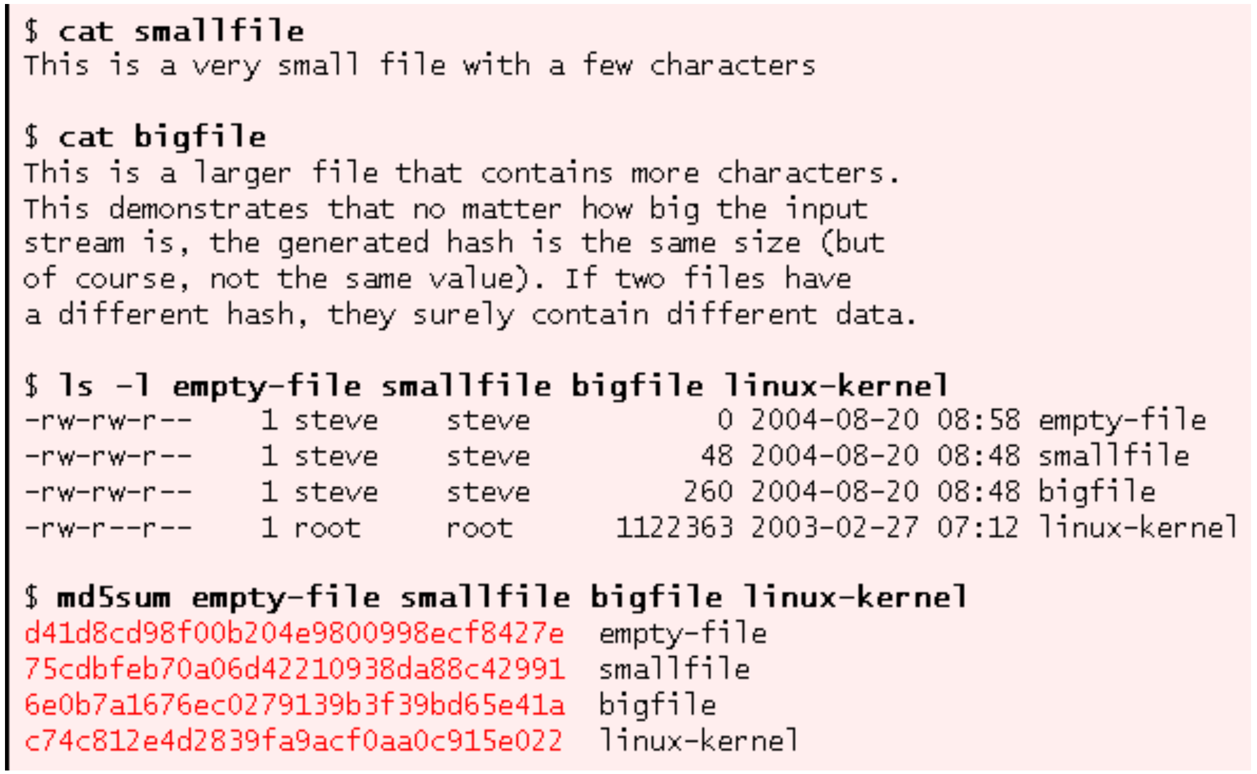
Subround example



- M_i = 32 bits from Y
- K_i = 32 bit constant (different for each round)
- Four functions F used in different rounds
    - $F(B,C,D) = (B \wedge C) \vee (\text{not}B \wedge D)$
    - $G(B,C,D) = (B \wedge D) \vee (C \wedge \text{not}D)$
    - $H(B,C,D) = B \oplus C \oplus D$
    - $I(B,C,D) = C \oplus (B \vee \text{not}D)$

Uses of MD5

- Many Unix and Linux systems provide the **md5sum** program; here, the "streams of data" are "files“
- This shows that *all* input streams yield hashes of the same length



```
$ cat smallfile
This is a very small file with a few characters

$ cat bigfile
This is a larger file that contains more characters.
This demonstrates that no matter how big the input
stream is, the generated hash is the same size (but
of course, not the same value). If two files have
a different hash, they surely contain different data.

$ ls -l empty-file smallfile bigfile linux-kernel
-rw-rw-r--    1 steve    steve           0 2004-08-20 08:58 empty-file
-rw-rw-r--    1 steve    steve          48 2004-08-20 08:48 smallfile
-rw-rw-r--    1 steve    steve         260 2004-08-20 08:48 bigfile
-rw-r--r--    1 root     root      1122363 2003-02-27 07:12 linux-kernel

$ md5sum empty-file smallfile bigfile linux-kernel
d41d8cd98f00b204e9800998ecf8427e  empty-file
75cdbfeb70a06d42210938da88c42991  smallfile
6e0b7a1676ec0279139b3f39bd65e41a  bigfile
c74c812e4d2839fa9acf0aa0c915e022  linux-kernel
```

Strength of MD5

- Try changing just one character of a small test file: even very small changes to the input yields sweeping changes in the value of the hash (avalanche effect).
- MD5 hash is dependent on all message bits
- Computational complexity of brute force attacks
  - to obtain 2 messages with same digest is 2^{64} op.
  - to find a message with a given digest is 2^{128} op.

Limits of MD5

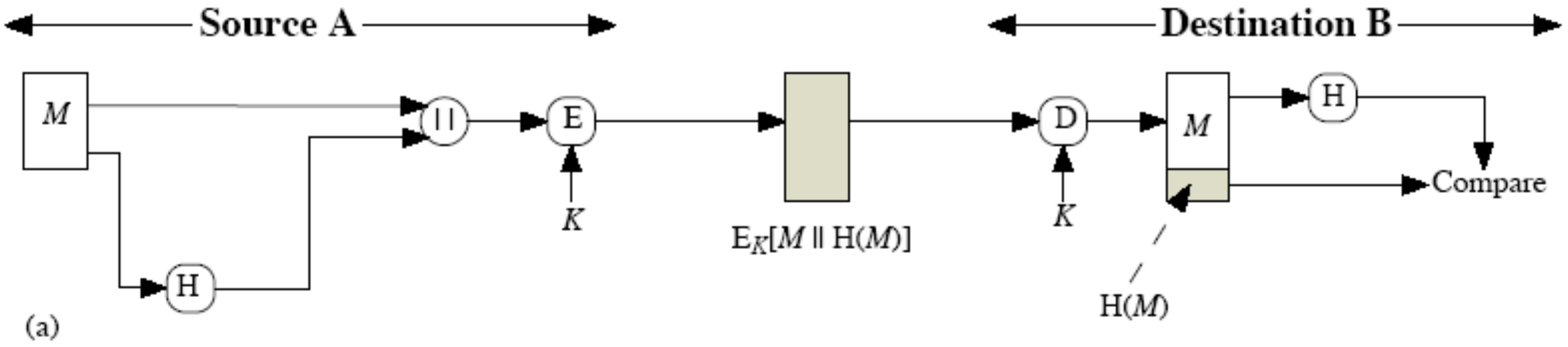
- We now know that MD5 is vulnerable
  - On March 2006, V. Klima published an algorithm that can find a collision within one minute on a single notebook computer
  - On the right, two strings producing an MD5 collision, that is the same MD5 hash 79054025255fb1a26e4bc422aef54eb4
  - http://www.mscs.dal.ca/~selinger/md5collision/

- d131dd02c5e6eec4693d9a0698aff95c
  2fcab5**8**712467eab4004583eb8fb7f89
  55ad340609f4b30283e488832**5**71415a
  085125e8f7cdc99fd91dbdf280373c5b
  d8823e3156348f5bae6dacd436c919c6
  dd53e2**b**487da03fd02396306d248cda0
  e99f33420f577ee8ce54b67080**a**80d1e
  c69821bcb6a8839396f965**2**b6ff72a70

- d131dd02c5e6eec4693d9a0698aff95c
  2fcab5**0**712467eab4004583eb8fb7f89
  55ad340609f4b30283e4888325**f**1415a
  085125e8f7cdc99fd91dbd7280373c5b
  d8823e3156348f5bae6dacd436c919c6
  dd53e2**3**487da03fd02396306d248cda0
  e99f33420f577ee8ce54b67080**2**80d1e
  c69821bcb6a8839396f965**a**b6ff72a70

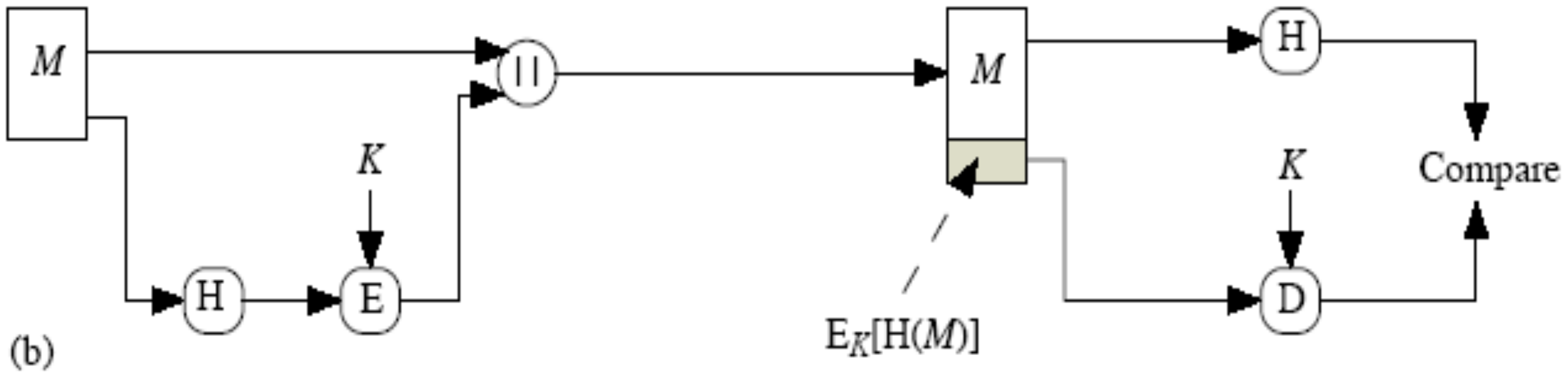
Uses of Hash Function / 1

- The hash code gives redundancy to check message authentication
- The encryption of M and hash gives secrecy and source authentication:
  - e.g.: the use of a symmetric key K allows sender identification



Uses of Hash Function / 2

- The hash code gives message authentication
- The encryption of the hash only allows sender authentication and fast computation
- $E_K[H(M)]$ is nothing but a MAC



Digital Signatures

- Message authentication does not tackle with the lack of trust between sender and recipient
- if John sends to Mary an authenticated msg, following disputes can arise:
  - Mary can generate a message & claim it was sent by John since she has the authentication key shared with John
  - John can deny to be the sender of a message since Mary can create a fake message, there is no way to demonstrate that the sender was really John
- Digital signatures have been proposed to avoid these kind of problems

Digital Signatures

- The purpose of a digital signature is to provide a means for an entity to bind its identity to a piece of digital information.
- The process of *signing* entails transforming the message and some secret information held by the entity into a tag called a *signature*.

Digital Signatures

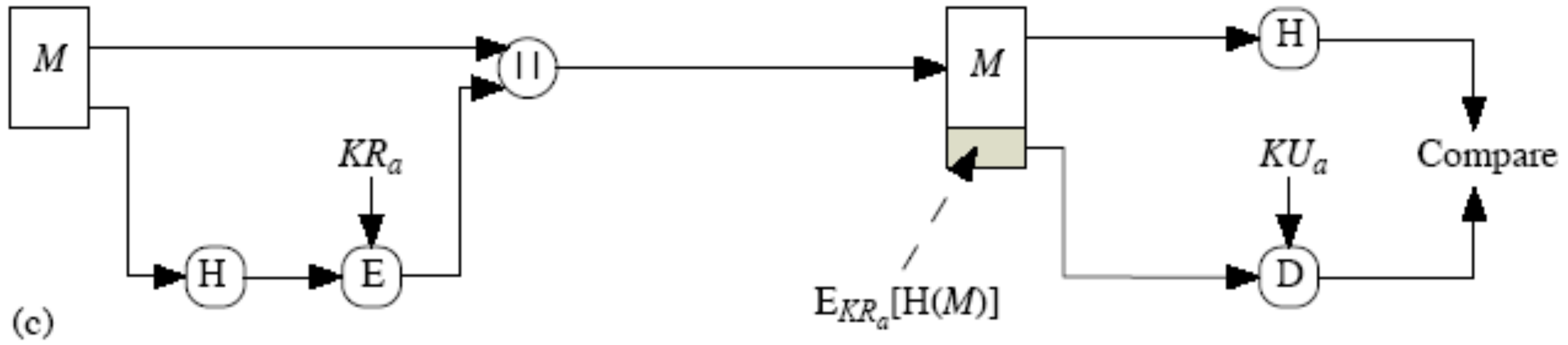
- Similar to the manual signatures
- Digital signatures provide the ability to:
  - verify author & time of signature
  - authenticate message contents
  - can be verified by third parties to resolve disputes
- They include authentication functionalities plus additional features

Digital Signature Properties

- Must depend on the message signed
- Must use information specific of sender
  - to prevent message forgery and repudiation
- Must be relatively easy to produce
- Must be relatively easy to recognize & verify
- Must be computationally infeasible to forge
  - a new message for existing digital signature
  - a fraudulent digital signature for given message
- Storing a copy of a digital signature should be practical

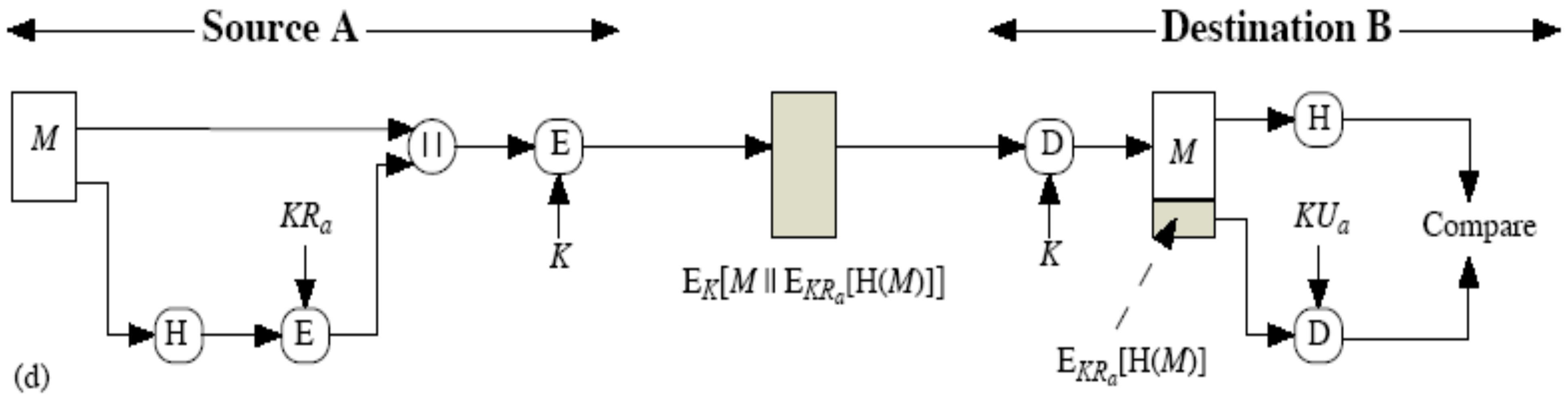
Signatures with Hash Function

- Relies on Public-key cryptographic
- The hash code gives message authentication
- E_{KRa} [H(M)] is a *digital signature*, assuring that the msg comes only from A
- No secrecy



Signature with Hash Function

- Like the previous method, with the addition of secrecy through symmetric algorithm
- ***Most used approach***

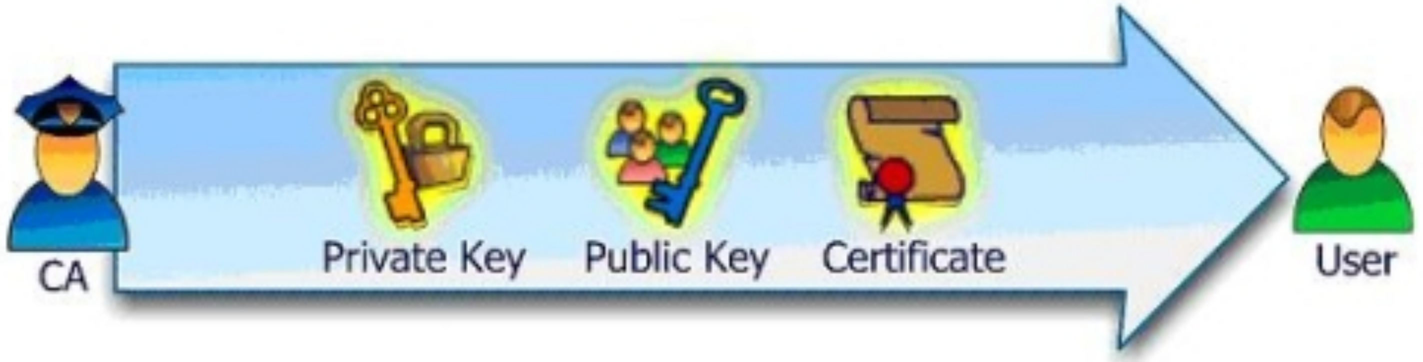


Signature with Hash Function

- There are several reasons to sign the hash instead of the whole document
- **Efficiency**
  - In this way the signature is much shorter. In addition hashing is generally much faster than encrypting.
- **Integrity**
  - Without the hash function, the text "to be signed" may have to be split (separated) in blocks small enough for the signature scheme to act on them directly.
  - However, the receiver of the signed blocks is not able to recognize if all the blocks are present and in the appropriate order.

Digital Signature Scheme

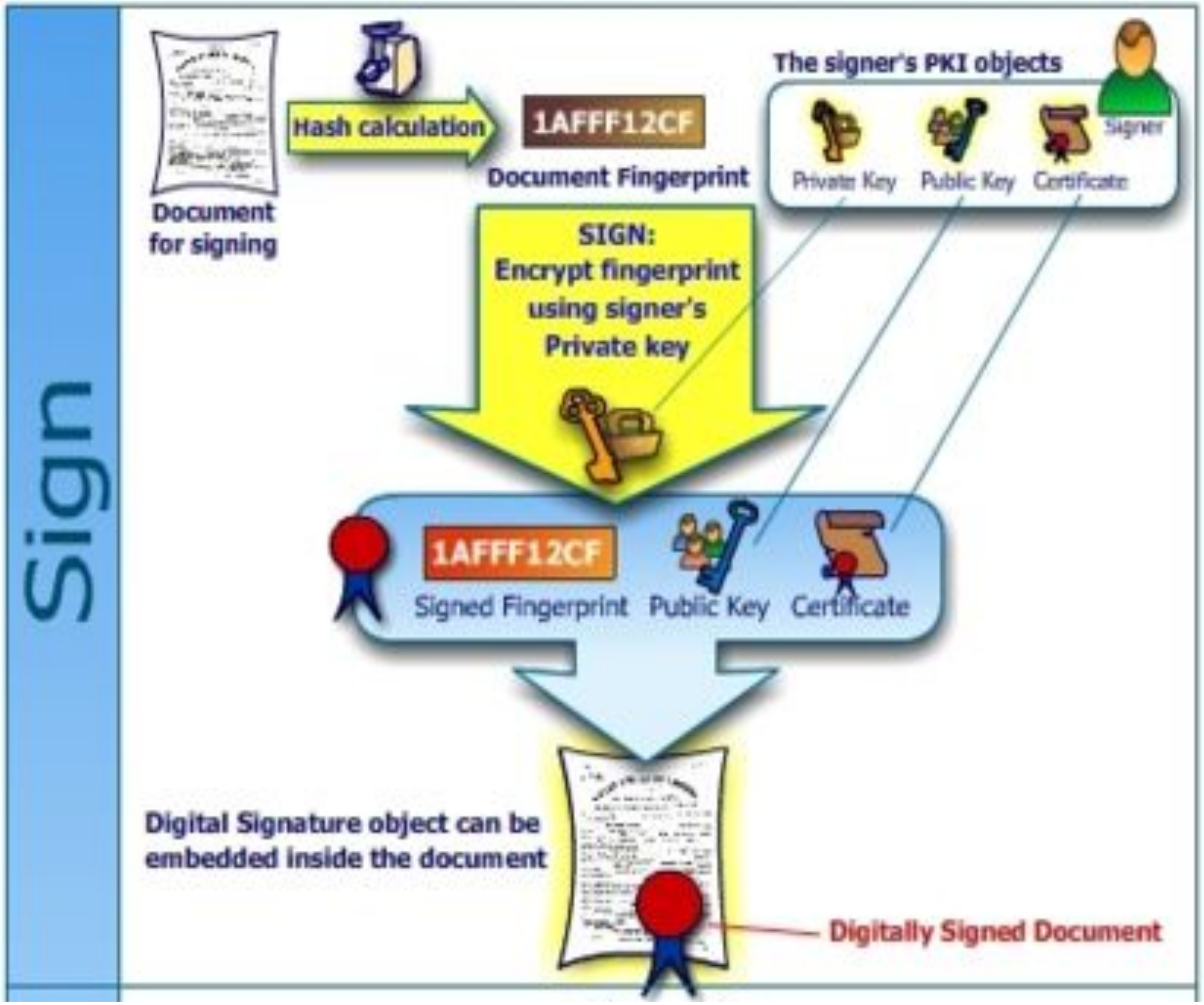
- It consists of 3 algorithms:
- A CA uses a key generation algorithm G to produce a "key pair" (PK, SK) for the signer.
- PK is the verifying key, which is to be public, and SK is the signing key, to be kept private.
- Authenticity of the key is ensured by a certificate



Digital Signature Scheme

A signing algorithm S,

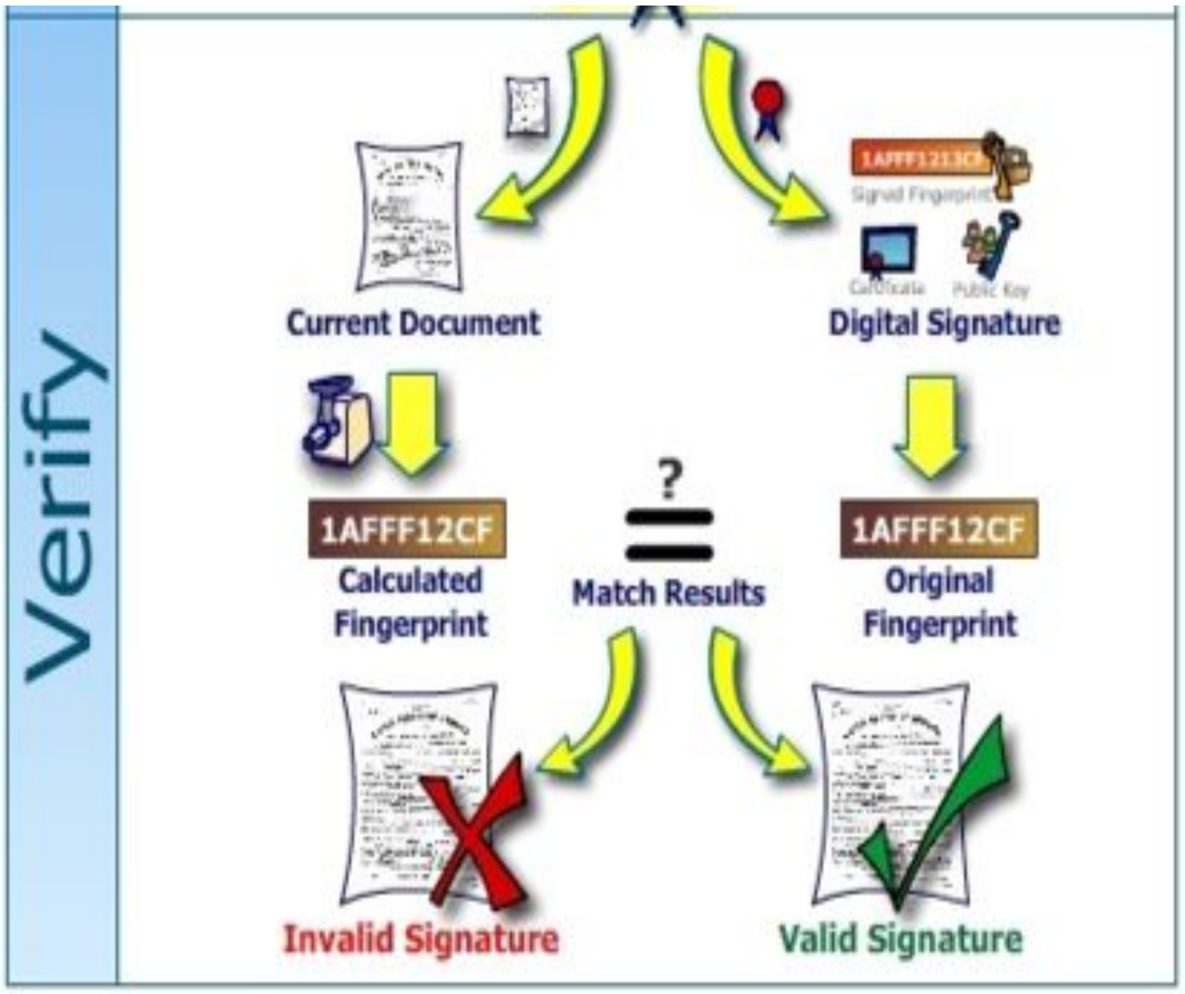
- Upon inputting a message m and a signing key SK
- It produces a signature σ.



Digital Signature Scheme

A *signature verifying* algorithm *V*, that

- Given a message *m*, a verifying key *PK*, and a signature σ,
- Accepts or reject the signed document

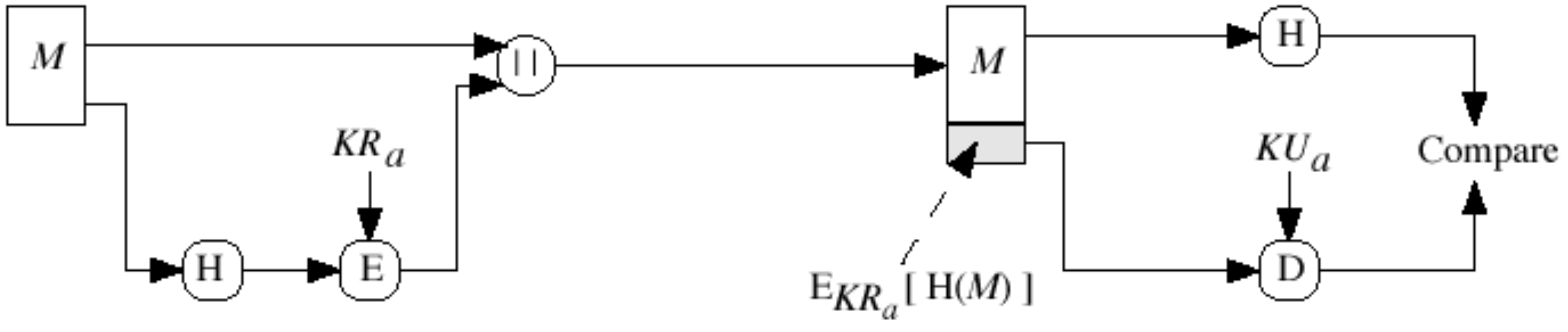


Digital Signature Scheme

- In the famous paper "New Directions in Cryptography", Diffie and Hellman first described the notion of a digital signature scheme, although they only conjectured that such schemes existed.
- Soon afterwards, Rivest, Shamir, and Adleman invented the RSA algorithm that could be used for primitive digital signatures.
- The first widely marketed software package to offer digital signature was Lotus Notes 1.0, released in 1989, which used the RSA algorithm.

Digital Signature Scheme with RSA

- In RSA the d.s. is the hash of M, encrypted with the sender's private key
- The signer computes σ = $H(M)^d$ mod n.
- To verify, the receiver checks that $σ^e$ = $H(M)$ mod n.



(a) RSA Approach

Digital Signature Standard (DSS)

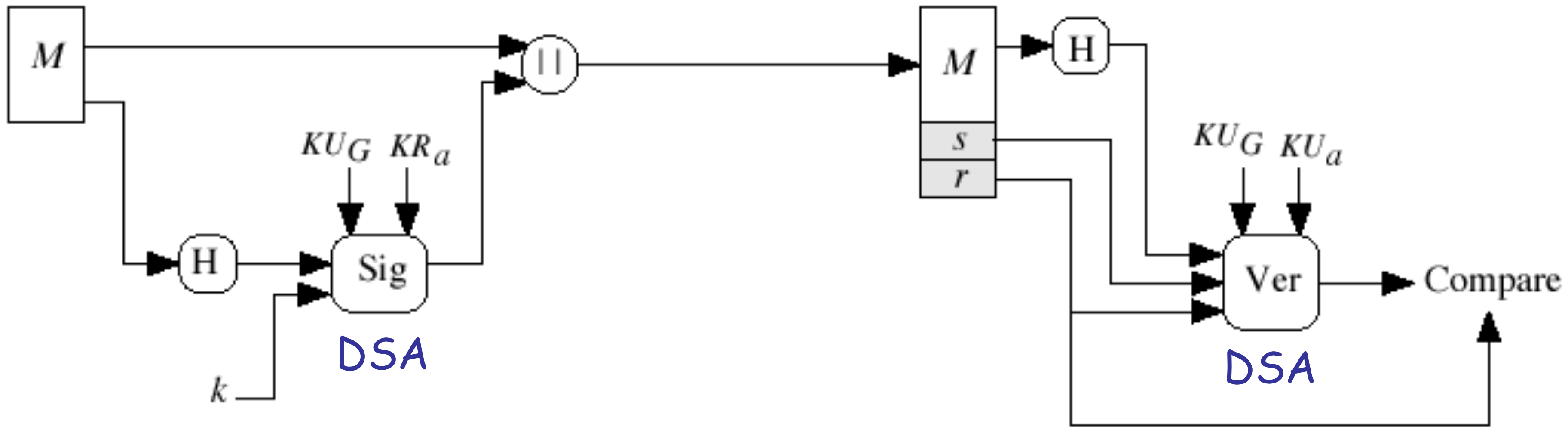
- The National Institute of Standards and Technology (NIST) in 1993 adopted their **Digital Signature Standard (DSS)**.

- It uses the SHA (Secure Hash Algorithm) to generate a hash, and a new digital signature algorithm, DSA (Digital Signature Algorithm).
- DSS is the standard, DSA is the algorithm

- DSS designed only for digital signatures, not for cryptography or key exchange.

DSS

- In DSS H(M) and a random number k are signed with sender's private key + a global public key.
- The d.s. is given by two components, s and r.
- Result of verification is compared only with r.



(b) DSS Approach

Digital Signature Algorithm (DSA)

- DSA creates a 320 bit signature, consisting of two 160-bit integers r and s.
  - The integer r is a function of a 160-bit random number k (ephemeral key) that changes with every message
  - The integer s is a function of: message, signer's private key x, integer r and ephemeral key k
- Security depends on difficulty of solving the discrete logarithm problem

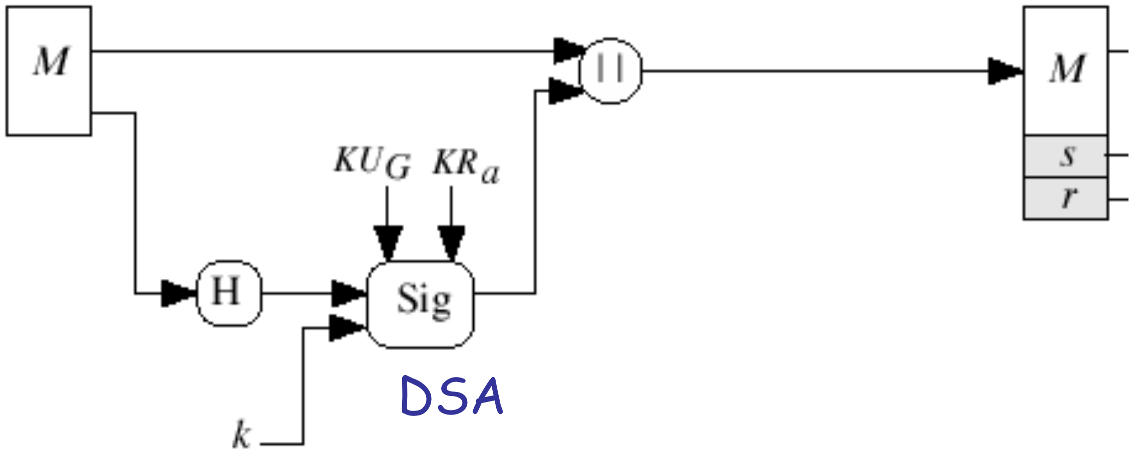
DSA Key generation

- Choose a 160-bit prime q.
- Choose an L-bit prime p, such that $p=qz+1$ for some integer z, $512 \leq L \leq 2048$, and L divisible by 64.
- Choose h, where $1 < h < p-1$ and let $g = h^z \bmod p > 1$.
- Randomly choose the private key x, $0 < x < q$.
- Calculate the public key $y = g^x = h^{zx} \bmod p$.

- $KR_a = x$
- $KU_a = y = h^{zx}$
- $KU_G = (p, q, g)$ can be shared between different users

DSA Signing

- Generate a random value k where $0 < k < q$
  - k is kept secret
- Calculate $r = (g^k \bmod p) \bmod q$
- Calculate $s = (k^{-1}(H(M) + xr)) \bmod q$
  - $H(M)$ is the SHA-1 hash function applied to message M
- Recalculate signature in the case that $r = 0$ or $s = 0$
- The signature is (r,s)



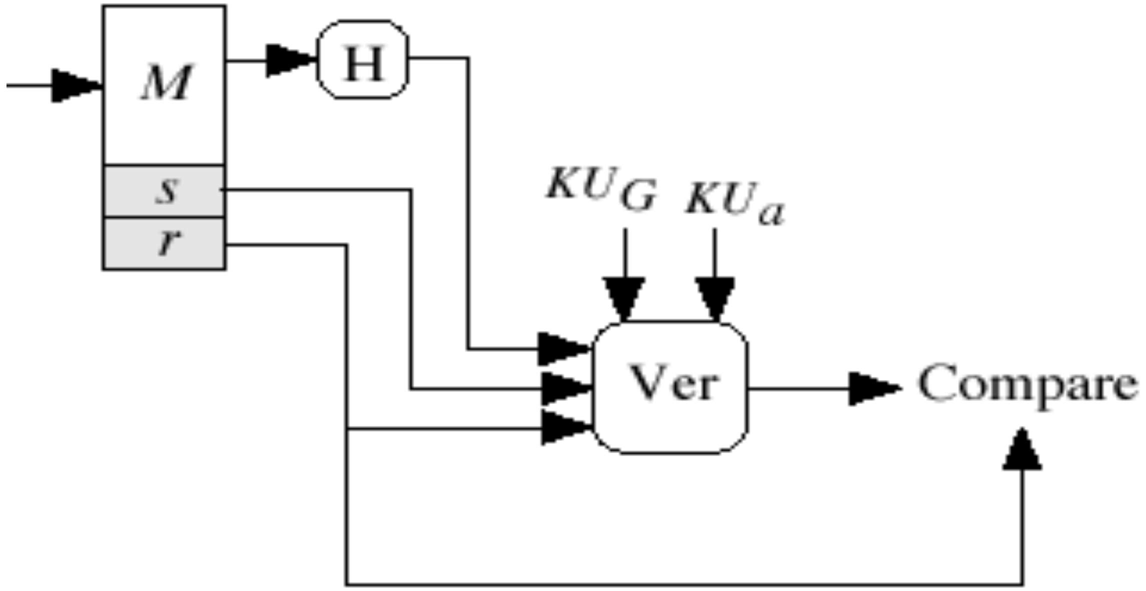
DSA Signature Verification

- To verify the signature, the receiver needs (**p**, **q**, **g**) and the sender's public key **y**.
- The parameters **p**, **q**, and **g** can be shared by many users.

$KR_a = x$

$KU_a = y$

$KU_G = p,g,q$



DSA Signature Verification

- Reject the signature if $0<r<q$ or $0<s<q$ not satisfied.
- Calculate $w = (s)^{-1} \bmod q$
- Calculate $u1 = (H(M)*w) \bmod q$
- Calculate $u2 = (rw) \bmod q$
- Calculate $v = ((g^{u1}y^{u2}) \bmod p) \bmod q$
- **The signature is valid if $v = r$**

DSA Signature Verification

- If **v = r**, then the signature is verified and the verifier can have high confidence that the received message was sent by the party holding the secret key **x** corresponding to **y**.
- If **v** does not equal **r**, the message is considered invalid.

Correctness of DSA

- From $g = h^z \bmod p$ it follows:
- $g^q \equiv h^{qz} \equiv h^{p-1} \equiv 1 \pmod p$ by Fermat's little theorem (p prime, $1<h<p-1$, $\gcd(h,p)=1$).

- Now let $m \bmod q = n \bmod q$, i.e. $m = n + kq$ for some integer k. Then:
- $g^m \bmod p = g^{n+kq} \bmod p = (g^n g^{kq}) \bmod p = ((g^n \bmod p)(g^q \bmod p)^k) \bmod p = g^n \bmod p$

Correctness of DSA

We want to verify that:

$v = ((g^{u1} * y^{u2}) \bmod p) \bmod q = r;$

$(g^{(H(M)*w) \bmod q} * y^{(r*w) \bmod q}) \bmod p =$

$= (g^{(H(M)*w) \bmod q} * g^{x(r*w) \bmod q}) \bmod p =$

$= (g^{(H(M)*w)} * g^{x(r*w)}) \bmod p =$

$= (g^{(H(M)+ xr)*w}) \bmod p = (g^{k}) \bmod p$

-> since $(g^{(a) \bmod q}) \bmod p = (g^{a+kq}) \bmod p = (g^{a}) \bmod p$ for previous slide

-> it follows from the defining equation of s:

$s = (k^{-1}(H(M) + x*r))$

$k = (H(M) + x*r)*(s^{-1}) = (H(M) + x*r)*w$

Correctness of DSA

We want to verify that:

$v = ((g^{u1} * y^{u2}) \bmod p) \bmod q = r;$

$(g^{(H(M)*w) \bmod q} * y^{(r*w) \bmod q}) \bmod p = (g^k) \bmod p$

So $v = ((g^k) \bmod p) \bmod q$

$r = (g^k \bmod p) \bmod q$

Then $v=r$

Random Numbers

- Random number generation is an important primitive in many cryptographic mechanisms.
- It has many uses in cryptography:
  - Session keys
  - Public key generation
  - Keystream for a one-time pad
- In all cases it is critical that these values be
  - Statistically random with uniform distribution, statistically independent
  - Unpredictable, cannot infer future sequence on previous values

Natural Random Noise

- Getting good random numbers is important but difficult !
- Best source is natural randomness in real world
  - find a regular but random event and monitor it
  - do generally need special hw to do this
  - e.g. radiation counters, radio noise, audio noise, thermal noise in diodes, etc
- Starting to see such hw in new CPU's

Natural Random Noise

- Since most *true sources* of random sequences (if there is such a thing) come from *physical means*, they tend to be either costly or slow in their generation.
- To overcome these problems, methods have been devised to construct *pseudorandom sequences* in a deterministic manner from a shorter random sequence called a *seed*.

Pseudorandom Number Generators

- Pseudorandom sequences appear to be generated by a truly random source to those who do not know how they are generated
- The generation algorithm is known, but the seed is not
- Many algorithms have been developed to generate pseudorandom bit sequences of various types.
  - Most of them are completely unsuitable for cryptographic purposes !

Pseudorandom Number Generators

- Minimum security requirement for a PRNG is that the random seed length k should be sufficiently large so that a search over 2^k possible seeds is infeasible
- Two general requirements:
  - the output sequences should be statistically indistinguishable from truly random sequences,
  - the output values should be unpredictable to an adversary with limited computational resources

Linear Congruential Generator

- The principle of a LCG is simple: a new pseudo random number X_n is generated on the basis of the previous one by adding a certain offset and wrapping the result if it exceeds a certain limit
- $X_{n+1} = (a + bX_n) \bmod c$
- where:
  - c (modulus) $c > 0$
  - b (multiplier) $0 < b < c$
  - a (increase) $0 \leq a < c$
  - X_0 (seed) $0 \leq X_0 < c$
  - if params are integers, X_n is an integer $0 \leq X_n < c$.

Linear Congruential Generator

- Randomness depends on the chosen values.
- The period of a general LCG is at most c, usually less.
- The LCG has a full period if and only if:
  - a and c are relatively prime,
  - (b-1) is divisible by all prime factors of c
  - (b-1) is a multiple of 4 if c is a multiple of 4.

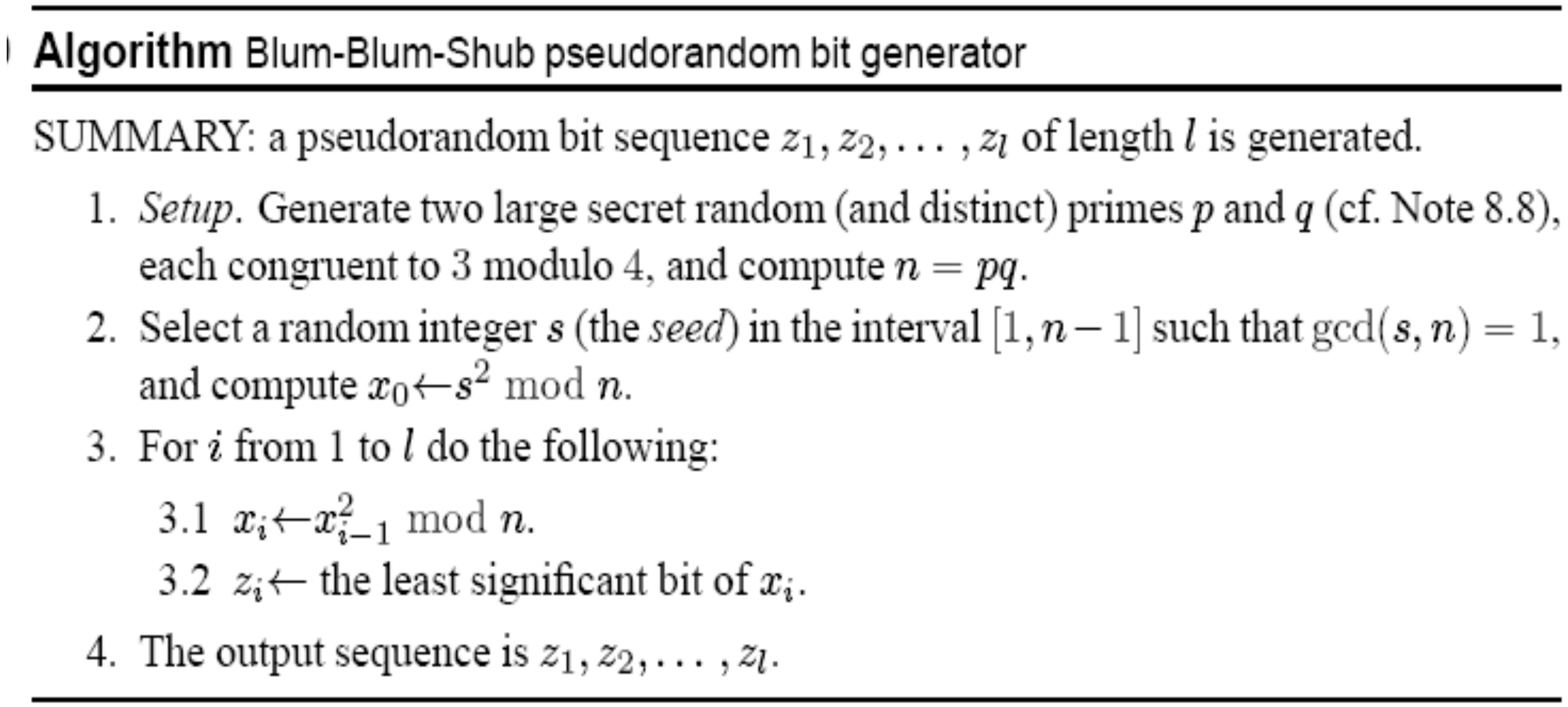
Linear Congruential Generator

- While LCG are commonly used for simulation purposes and probabilistic algorithms, they are *predictable* and hence entirely insecure:
- Given a partial output sequence, the remainder of the sequence can be reconstructed even if the parameters a, b, and c are unknown.

Blum Blum Shub Generator

- The Blum-Blum-Shub pseudorandom bit generator is a cryptographically secure pseudorandom bit generator (CSPRBG) under the assumption that integer factorization is intractable.
- It forms the basis for the Blum-Goldwasser probabilistic public-key encryption scheme

Blum Blum Shub Generator



Algorithm Blum-Blum-Shub pseudorandom bit generator

SUMMARY: a pseudorandom bit sequence $z_1, z_2, \ldots, z_l$ of length l is generated.

```
1. Setup. Generate two large secret random (and distinct) primes $p$ and $q$ (cf. Note 8.8),
   each congruent to 3 modulo 4, and compute $n = pq$.
2. Select a random integer $s$ (the seed) in the interval $[1, n-1]$ such that $\gcd(s, n) = 1$,
   and compute $x_0 \leftarrow s^2 \bmod n$.
3. For $i$ from 1 to $l$ do the following:
   3.1 $x_i \leftarrow x_{i-1}^2 \bmod n$.
   3.2 $z_i \leftarrow$ the least significant bit of $x_i$.
4. The output sequence is $z_1, z_2, \ldots, z_l$.
```

p mod 4 = q mod 4 = 3 ensures that the cycle length is large

Blum Blum Shub Generator

- Based on public key algorithms
- Unpredictable, passes **next-bit** test
- Security rests on difficulty of factoring *n*
- Slow, since very large numbers must be used
- Too slow for cipher use, good for key generation

References

- W. Stallings, Cryptography and Network Security, Mc Graw Hill
  - (chapters 7,11,12,13)
- A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press
  - (chapter 1)
- Wikipedia, the free encyclopedia