Key Exchange and Hashing



Dr. Lokesh Chouhan

Associate Professor

गृह मंत्रालय
MINISTRY OF
HOME AFFAIRS
सत्यमेव जयते

राष्ट्रीय न्यायालयिक विज्ञान विश्वविद्यालय
(राष्ट्रीय महत्त्व का संस्थान, गृह मंत्रालय, भारत सरकार)
National Forensic Sciences University
(An Institution of National Importance under Ministry of Home Affairs, Government of India)

NFSU
विद्यया अमृतं अश्नुते

E-Mail: Lokeshchouhan@gmail.com, Lokesh.chouhan_goa@nfsu.ac.in

Mob: +91-898924399, 9827235155

Unit 4

Public Key Cryptography-IV

Key Exchange

- Public key systems are much slower than private key system
  - Public key system is then often for short data
    - Signature, key distribution
- Key distribution
  - One party chooses the key and transmits it to other user
- Key agreement
  - Protocol such two parties jointly establish secret key over public communication channel
  - Key is the function of inputs of two users

Distribution of Public Keys

- can be considered as using one of:
  - Public announcement
  - Publicly available directory
  - Public-key authority
  - Public-key certificates

Public Key Management

- Simple one: publish the public key
  - Such as newsgroups, yellow-book, etc.
  - But it is not secure, although it is convenient
    - Anyone can forge such a announcement
    - Ex: user B pretends to be A, and publish a key for A
    - Then all messages sent to A, readable by B!
- Let trusted authority maintain the keys
  - Need to verify the identity, when register keys
  - User can replace old keys, or void old keys

Possible Attacks

- Observe all messages over the channel
  - So assume that all plaintext messages are available to all
- Save messages for reuse later
  - So have to avoid replay attack
- Masquerade various users in the network
  - So have to be able to verify the source of the message

Public Announcement

- users distribute public keys to recipients or broadcast to community at large
  - eg. append PGP keys to email messages or post to news groups or email list
- major weakness is forgery
  - anyone can create a key claiming to be someone else and broadcast it
  - until forgery is discovered can masquerade as claimed user

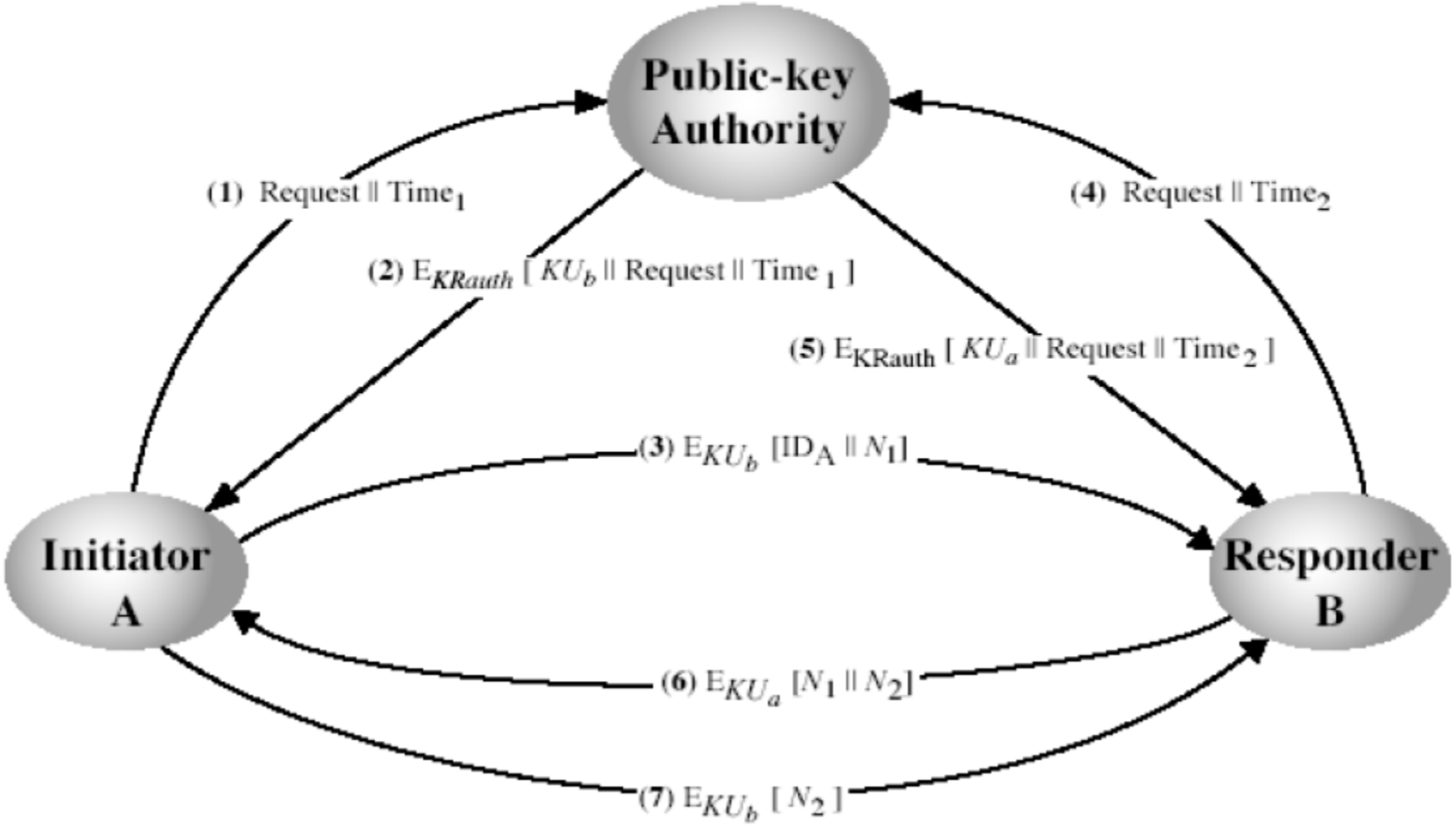
Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- directory must be trusted with properties:
  - contains {name,public-key} entries
  - participants register securely with directory
  - participants can replace key at any time
  - directory is periodically published
  - directory can be accessed electronically
- still vulnerable to tampering or forgery

Public-Key Authority

- improve security by tightening control over distribution of keys from directory
- has properties of directory
- and requires users to know public key for the directory
- then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed

Public-Key Authority



Cont.

- More advanced distribution
  - A sends request-for-key(B) to authority with time-stamp, that is, Ida|Idb|Time
  - Authority replies with key(B) (encrypted by its private key), that is E_{KTta}(KUb| Ida|Idb|Time)
  - A initiates a message to B, including a random number N_a, its ID_A
  - B then ask authority to get key(A)
  - B sends A (encrypted by A's public key) N_a and N_b
  - A then replies B N_b encrypted by B's public key

Cont.

- In above scheme, the authority is bottleneck
- New approach: certificate
  - Any user can read certificate, determine name and public key of the certificate's owner
  - Any user can verify the authority of certificate
  - Only the authority can create and update certificate
  - Any user can verify the time-stamp of certificate
- The certificate is
  - $C_A = E_{KR_{auth}}[T, ID_A, KU_A]$
  - Time-stamp is to avoid reuse of voided key

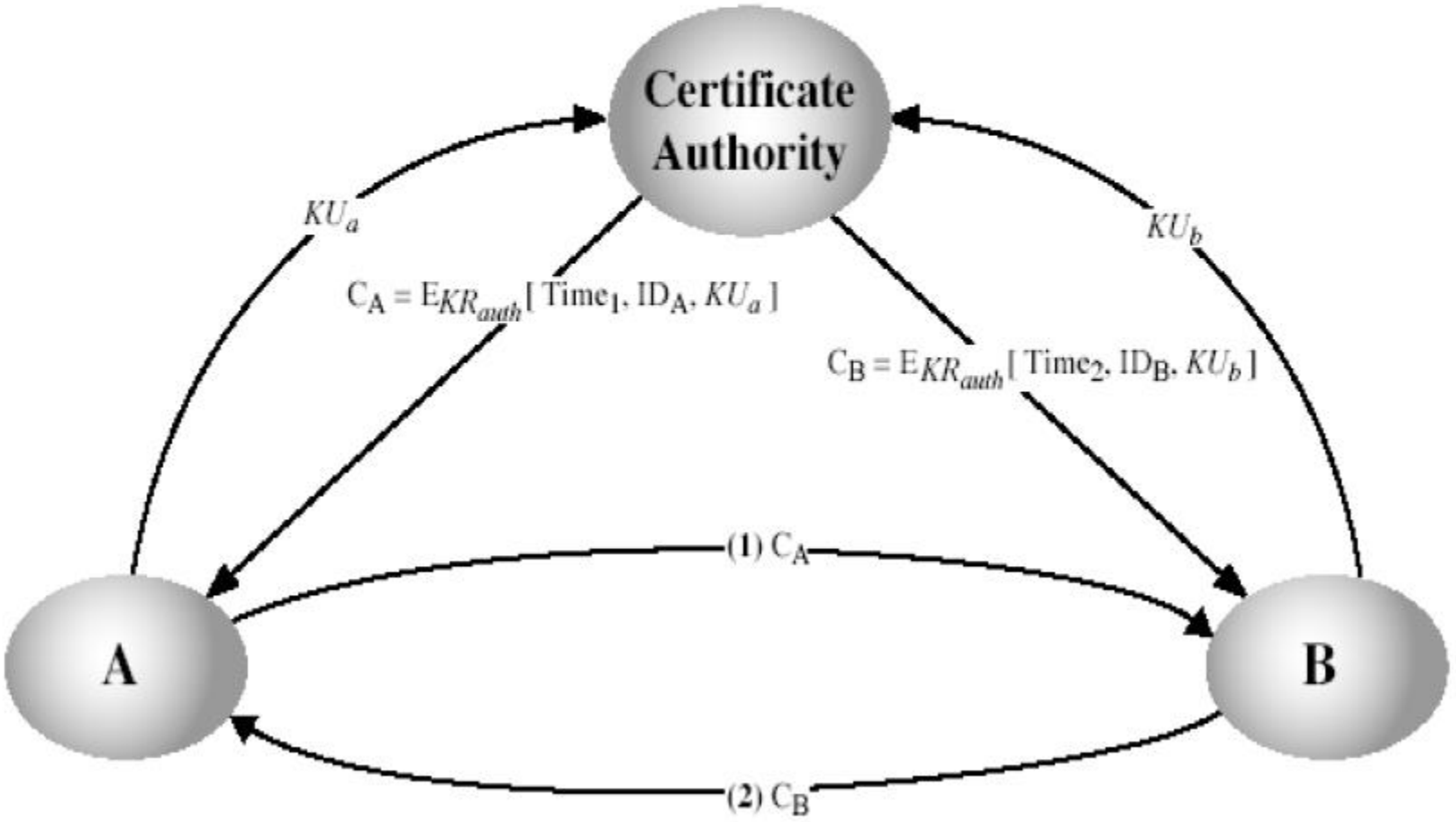
Public-Key Certificates

- certificates allow key exchange without real-time access to public-key authority
- a certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use etc
- with all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authorities public-key
- To validate the certificate, we need another certificate, one that matches the Issuer (of CA) in the first certificate. Then we take the RSA public key from the second (CA) certificate, use it to decode the signature on the first certificate to obtain an MD5 hash, which must match an actual MD5 hash computed over the rest of the certificate.

X.509

- The structure of a X.509 v3 digital certificate is as follows:
- Certificate
  - Version
  - Serial Number
  - Algorithm ID
  - Issuer
  - Validity
    - Not Before
    - Not After
  - Subject
  - Subject Public Key Info
    - Public Key Algorithm
    - Subject Public Key
  - Issuer Unique Identifier (Optional)
  - Subject Unique Identifier (Optional)
  - Extensions (Optional)
    - ...
- Certificate Signature Algorithm
- Certificate Signature

Public-Key Certificates



Message Authentication Digital Signature

- Authentication
  - Authentication requirements
  - Authentication functions
- Mechanisms
  - MAC: message authentication code
  - Hash functions, security in hash functions
  - Hash and MAC algorithms
    - MD5, SHA, RIPEMD-160, HMAC
- Digital signatures

Message Attacks

- Possible attacks
  - Disclosure
  - Traffic analysis
  - Masquerade
  - Content modification
  - Sequence modification
  - Time modification
  - Repudiation
    - Denial of the receipt of message by the destination or
    - Denial of the transmitting by the source

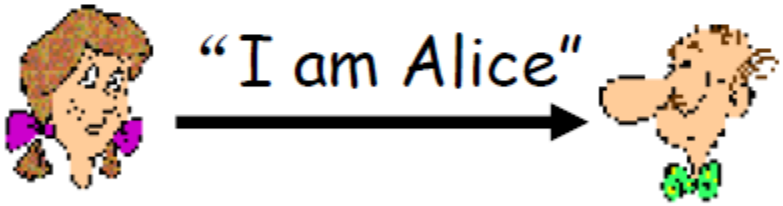
Authentication

- Enables receiver to verify message authenticity
  - Using some lower level functions as primitive
- Three types of functions
  - Message encryption
  - Message authentication code (MAC)
  - Hash function

Authentication

Goal: Bob wants Alice to "prove" her identity to him

Protocol ap1.0: Alice says "I am Alice"

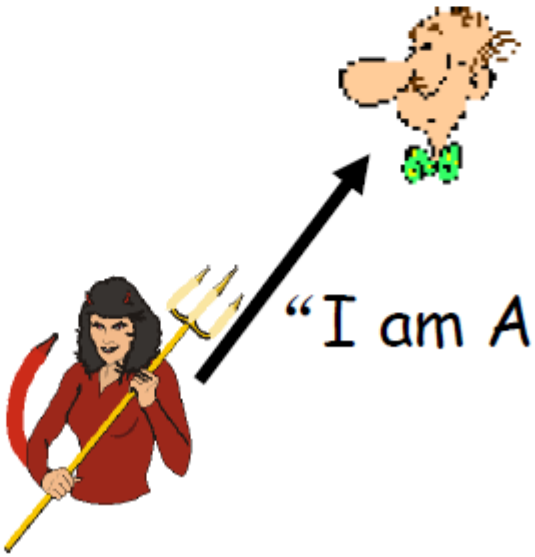


Failure scenario??

Authentication

Goal: Bob wants Alice to "prove" her identity to him

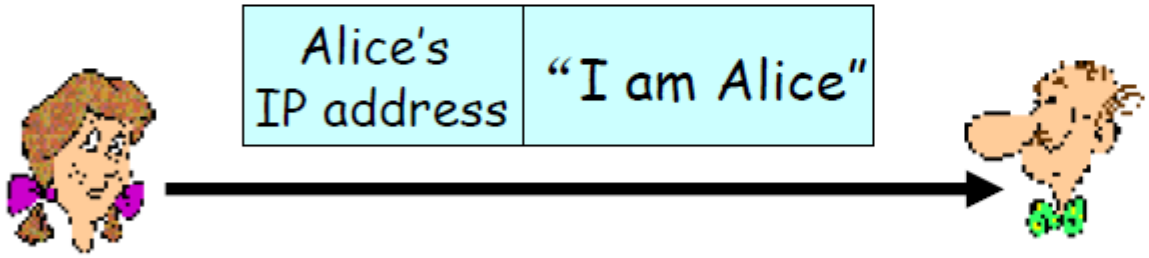
Protocol ap1.0: Alice says "I am Alice"



in a network, Bob can not "see" Alice, so Trudy simply declares herself to be Alice

Authentication: another try

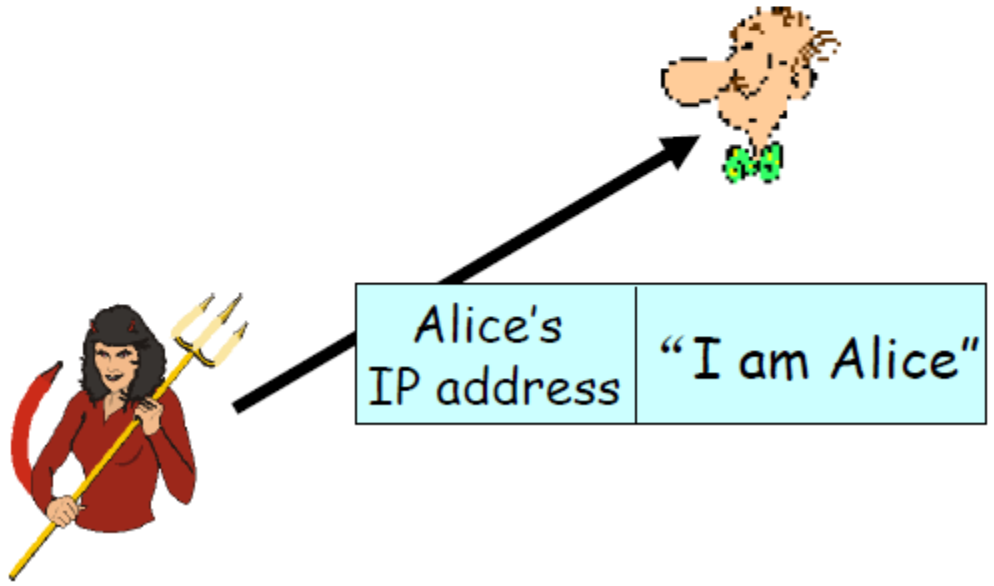
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Failure scenario??

Authentication: another try

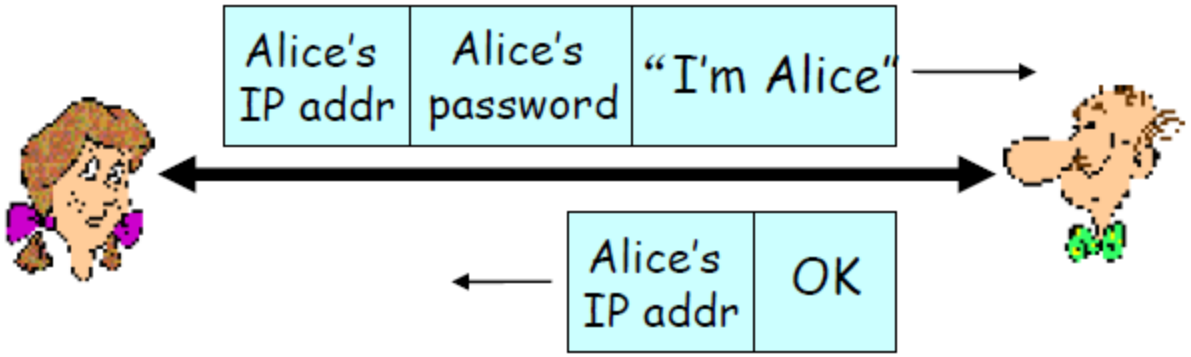
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Trudy can create a packet "spoofing" Alice's address

Authentication: another try

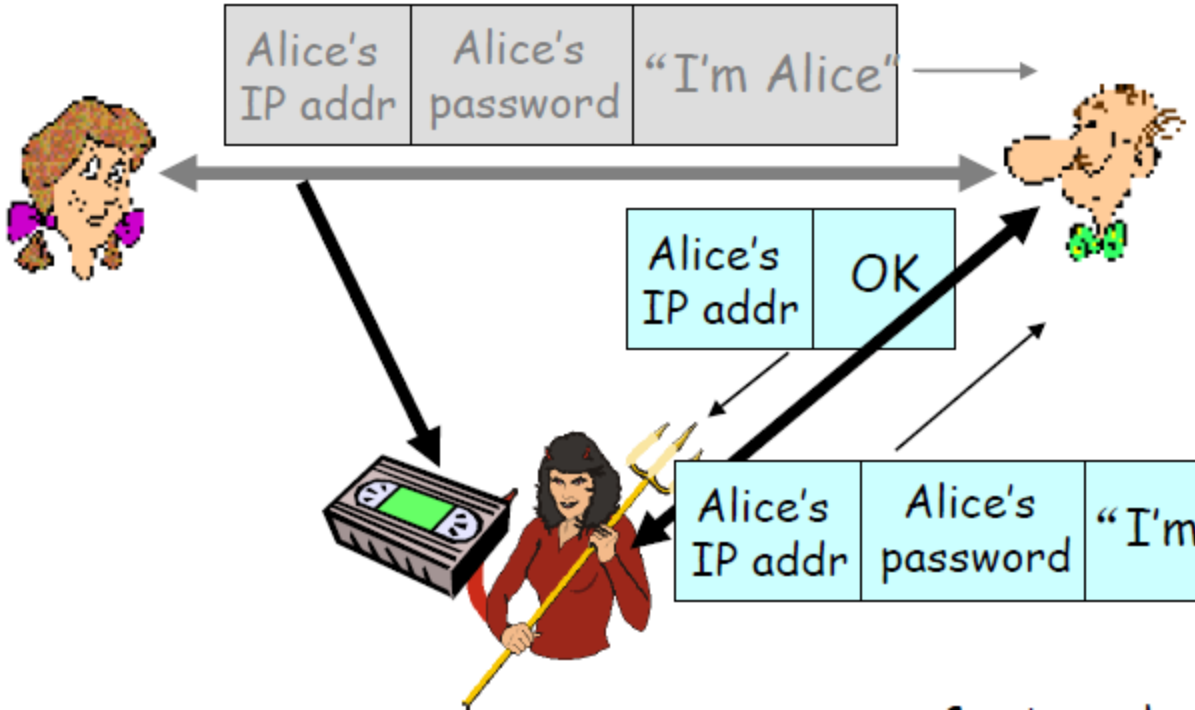
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

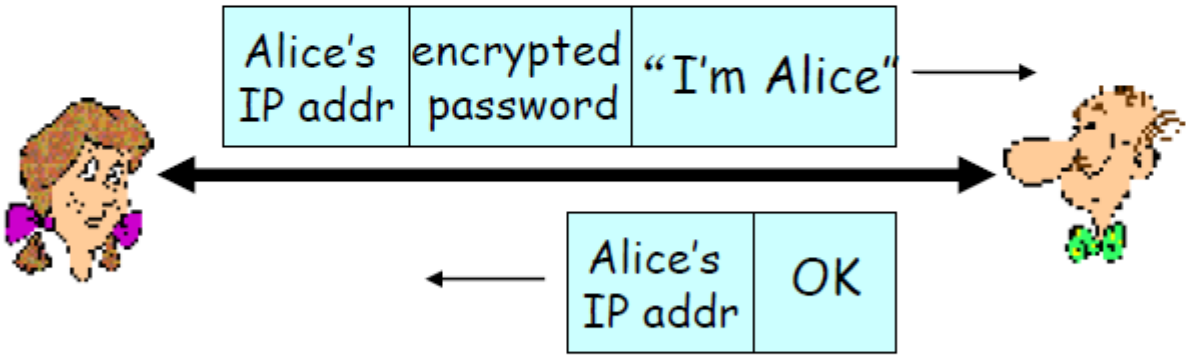
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: yet another try

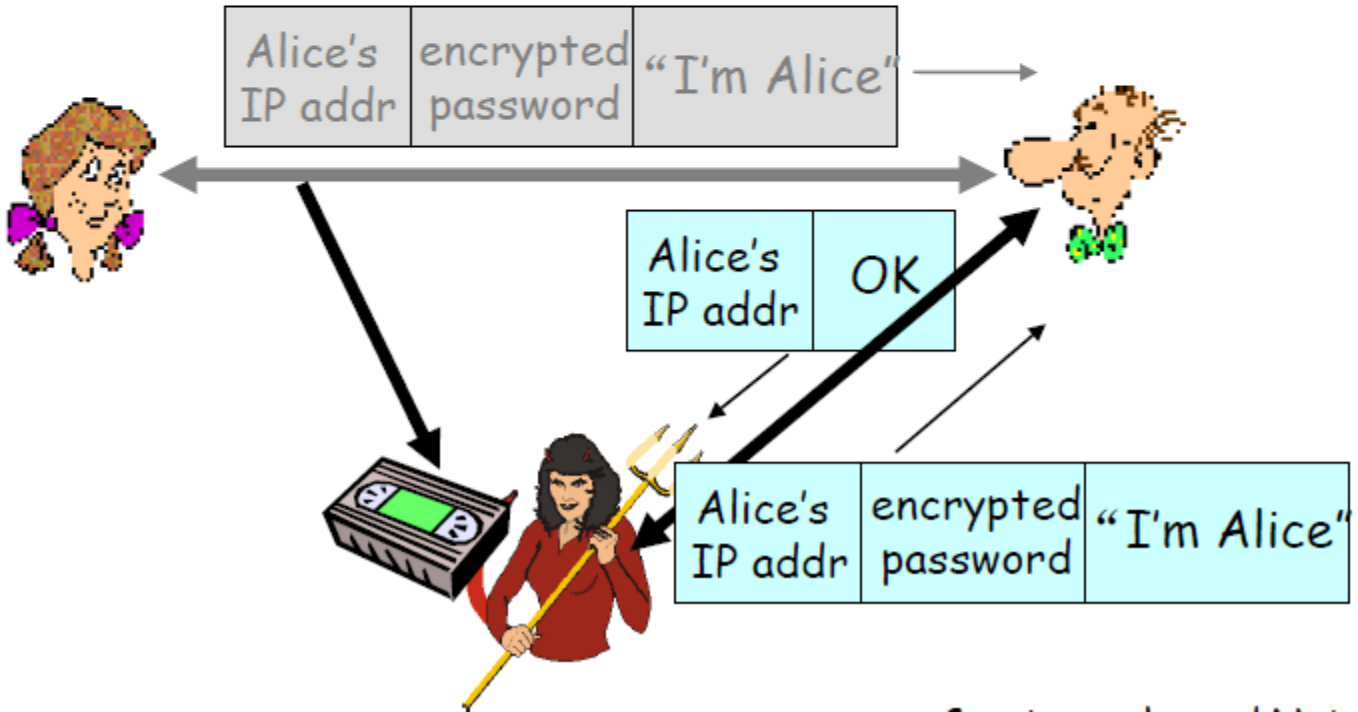
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



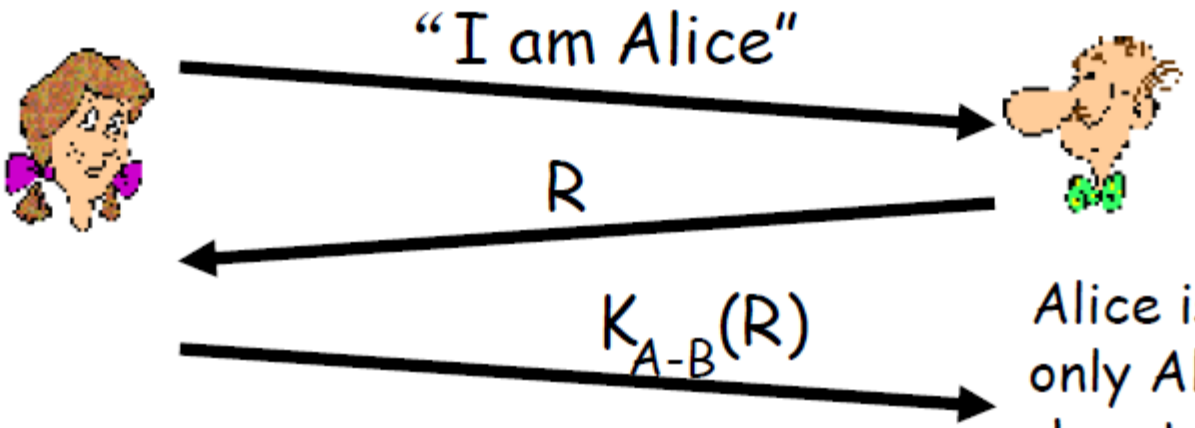
record and playback still works!

Authentication: yet another try

Goal: avoid playback attack

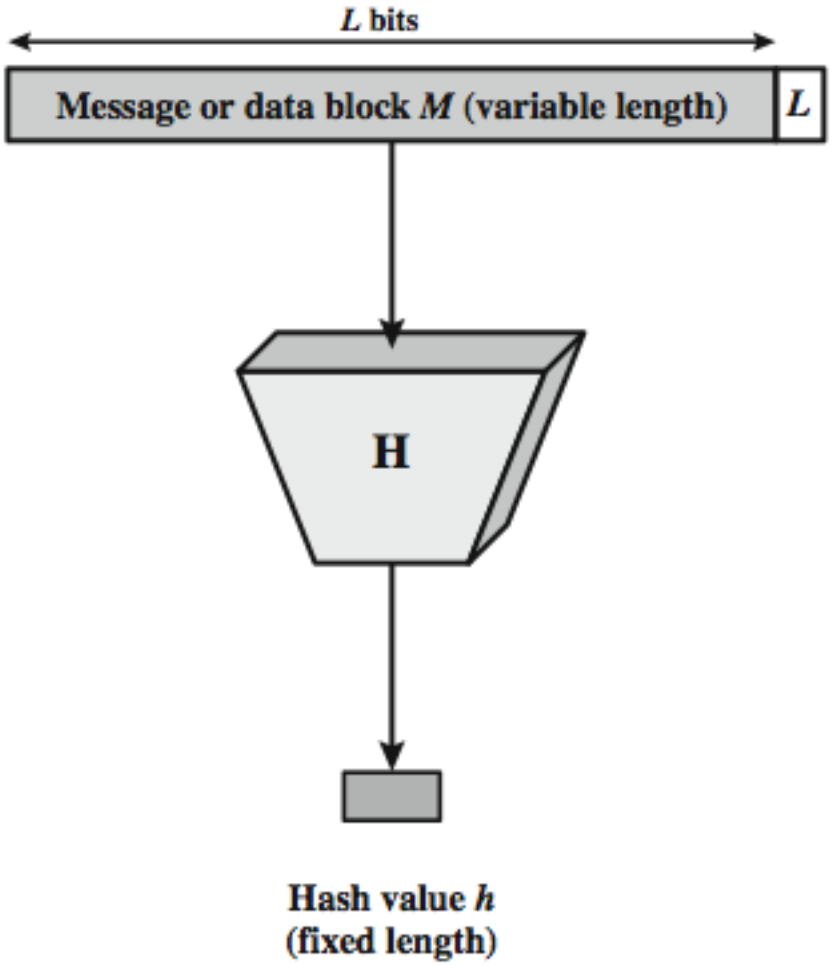
Nonce: number (R) used only *once -in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key



drawbacks?

Cryptographic Hash Function



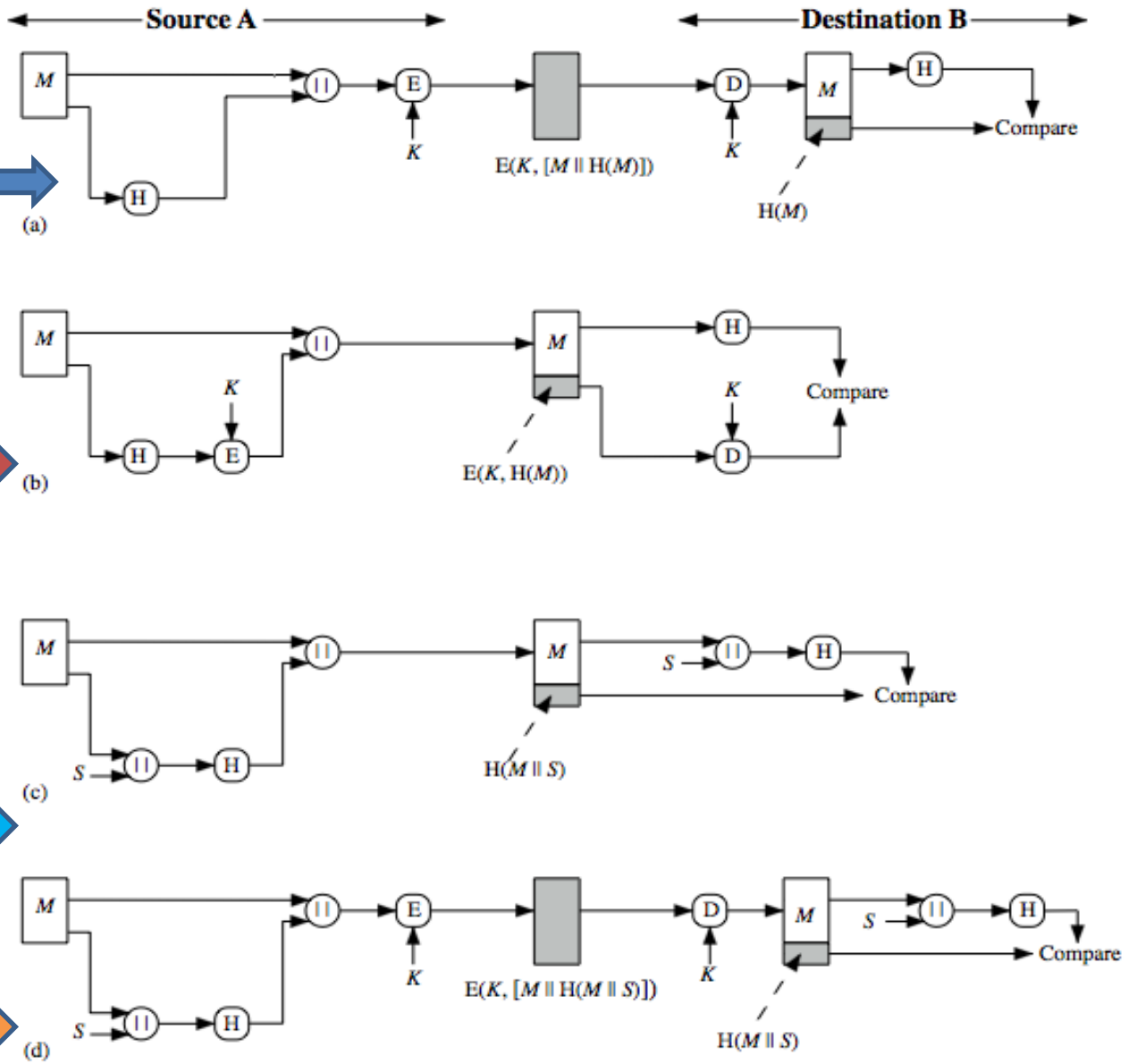
Hash Functions & Message Authentication

(a) The **message plus concatenated hash** code is **encrypted** using symmetric encryption.

(b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications not requiring **confidentiality.**

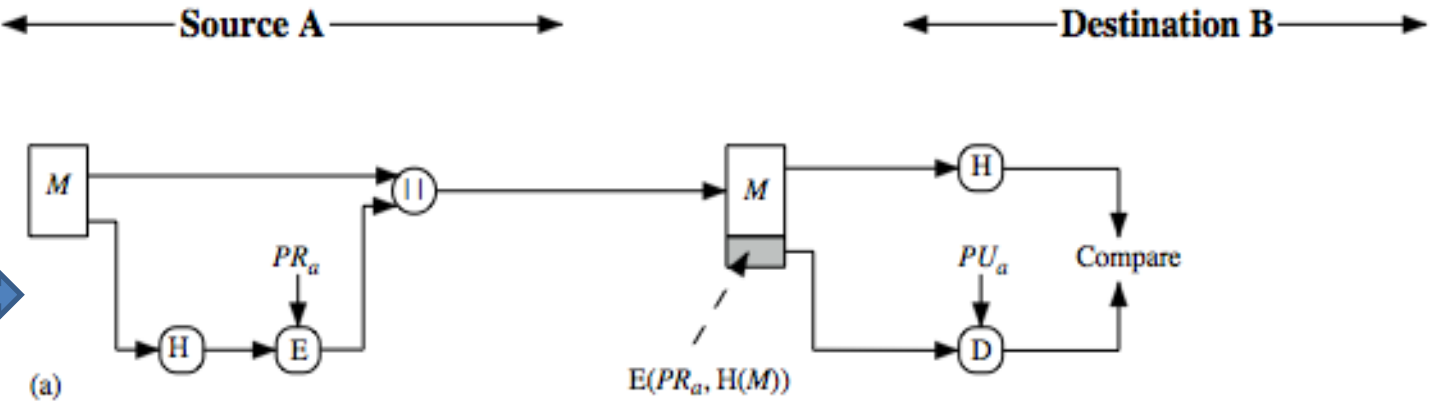
(c) The technique assumes that the two communicating parties share a common **secret value S.** A computes the hash value over the concatenation of M and S and appends the resulting hash value to M.

(d) **Confidentiality** can be added to the approach of (c) by encrypting the entire message plus the hash code.



Hash Functions & Digital Signatures

digital signature



a.The hash code is encrypted, using public-key encryption and using the sender's private key. This provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

digital signature + Encryption

b. If **confidentiality** as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

Other Hash Function Uses

- to create a one-way **password file**
  - store hash of password not actual password
- for **intrusion detection and virus detection**
  - keep & check hash of files on system
- **pseudorandom function (PRF) or pseudorandom number generator (PRNG)**

Two Simple Insecure Hash Functions

- consider two simple insecure hash functions
- **bit-by-bit exclusive-OR (XOR)** of every block
  - $C_i = b_{i1}$ *xor* b_{i2} *xor . . . xor* b_{im}
  - a longitudinal redundancy check
  - reasonably effective as data integrity check
- **one-bit circular shift on hash value**
  - for each successive *n-bit* block
    - rotate current hash value to left by1bit and XOR block
  - good for data integrity but useless for security

Hash Function Requirements

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that H(y) = h.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find y ! x with H(y) = H(x).
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).
Pseudorandomness	Output of H meets standard tests for pseudorandomness

Attacks on Hash Functions

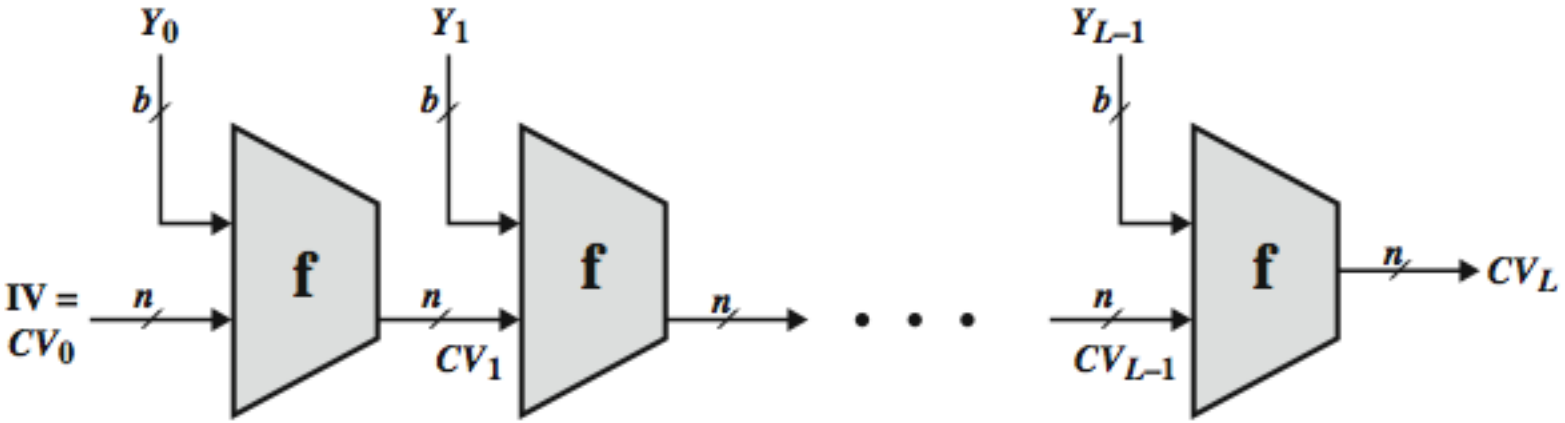
- have **brute-force** attacks and cryptanalysis
- a preimage or second preimage attack
  - find y s.t. $H(y)$ equals a given hash value
- collision resistance
  - find two messages x & y with same hash so $H(x) = H(y)$
- hence value $2^{m/2}$ determines strength of hash code against brute-force attacks
  - **128**-bits inadequate, **160**-bits suspect

Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - given user prepared to sign a valid message x
  - opponent generates **$2^{m/2}$ variations x' of x**, all with essentially the same meaning, and saves them
  - opponent generates $2^{m/2}$ variations y' of a desired fraudulent message y
  - two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
  - have user sign the valid message, then substitute the forgery which will have a valid signature
- conclusion is that need to use **larger MAC/hash**

Hash Function Cryptanalysis

- cryptanalytic attacks exploit some property of alg so faster than exhaustive search
- hash functions use iterative structure
  - process message in blocks (incl length)
- attacks focus on **collisions in function f**

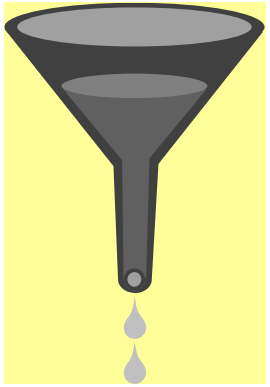


Block Ciphers as Hash Functions

- can use block ciphers as hash functions
  - using $H_0=0$ and zero-pad of final block
  - compute: $H_i = E_{M_i}[H_{i-1}]$
  - and use final block as the **hash value**
  - similar to **CBC but without a key**
- resulting hash is too small (64-bit)
  - both due to direct birthday attack
  - and to "meet-in-the-middle" attack
- other variants also susceptible to attack

MD5: Message Digest Version 5

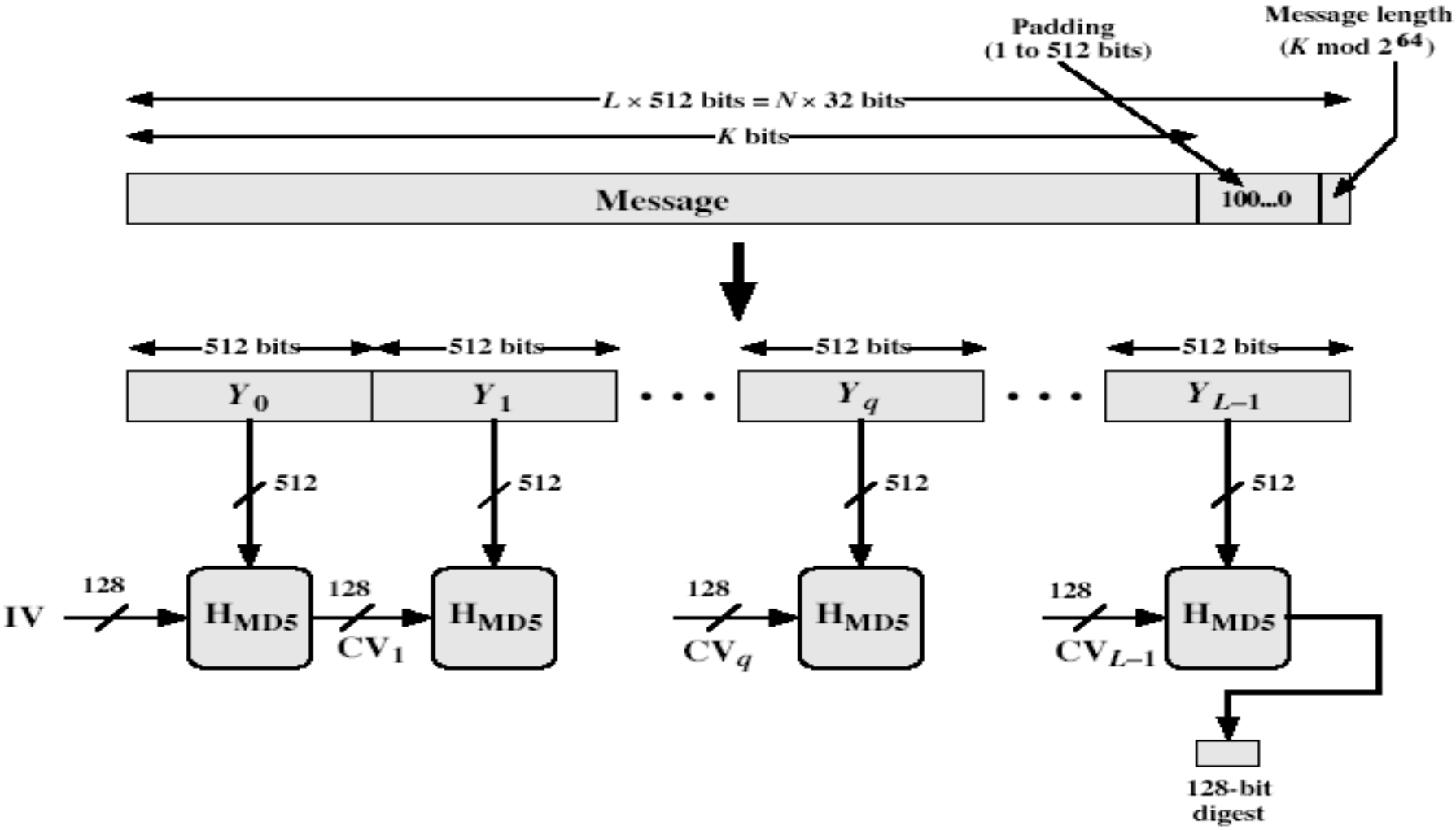
Input Message



Output 128 bits Digest

- Until recently the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns
- Specified as Internet standard RFC1321

MD5 Overview



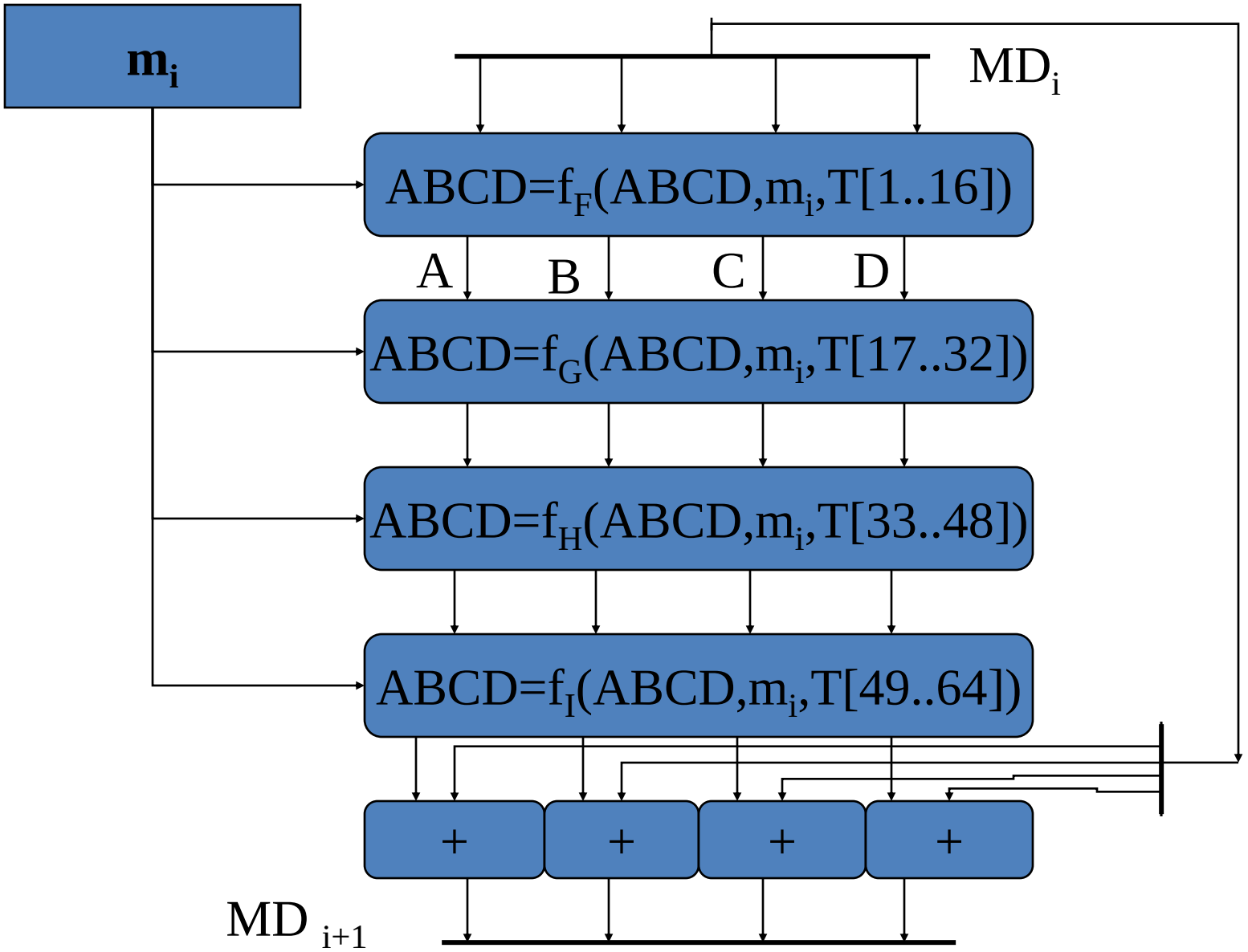
Padding Twist

- Given original **message M**, add padding bits "10^*" such that resulting length is 64 bits less than a multiple of 512 bits.
- Append (*original length in bits* mod 2^{64}), represented in 64 bits to the padded message
- Final message is chopped 512 bits a block

MD5 Process

- As many stages as the number of 512-bit blocks in the final padded message
- **Digest**: **Four**, **32**-bit words: **MD=A|B|C|D**
- Every message block **contains 16**, 32-bit words: $m_0|m_1|m_2...|m_{15}$
  - Digest MD_0 initialized to: A=01234567,B=89abcdef,C=fedcba98, D=76543210
  - Every stage consists of 4 passes over the message block, each modifying MD
- Each block 4 rounds, each round 16 steps

Processing of Block m_i - 4 Passes



Different Passes...

Each step *t* (0 <= *t* <= 79):

- **Input:**
  - m_t – a 32-bit word from the message

    With different shift every round
  - T_t – int(2^{32} * abs(sin(i))), 0<i<65

    Provided a randomized set of 32-bit patterns, which eliminate any regularities in the input data
  - ABCD: current MD
- **Output:**
  - ABCD: new MD

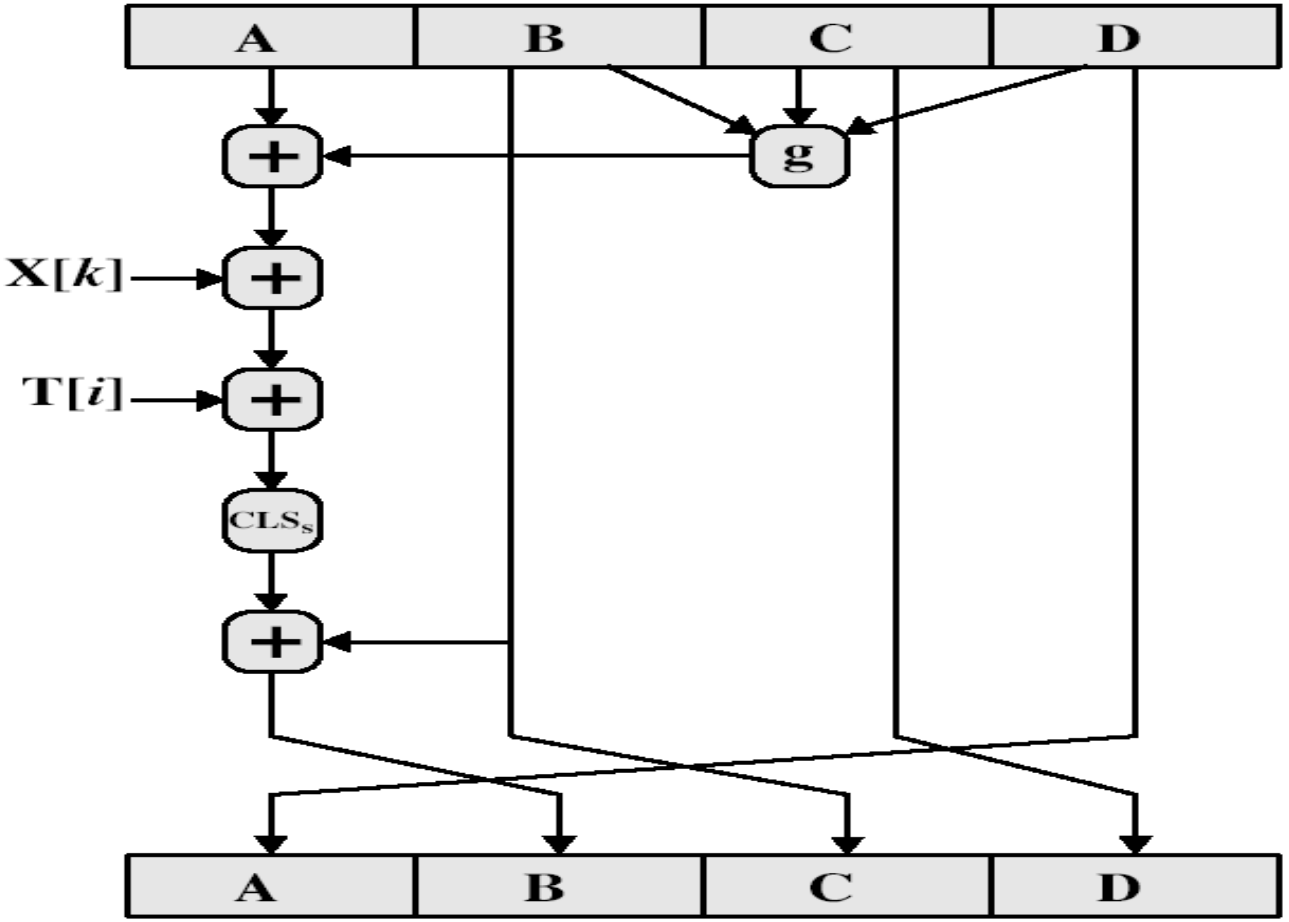
MD5 Compression Function

- Each round has 16 steps of the form:

```
a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)
```

- a,b,c,d refer to the 4 words of the buffer, but used in varying permutations
  - note this updates 1 word only of the buffer
  - after 16 steps each word is updated 4 times
- where **g(b,c,d)** is a different nonlinear function in **each round (F,G,H,I)**

MD5 Compression Function



Functions and Random Numbers

- F(x,y,z) == (x∧y)∨(~x ∧ z)
  – selection function
- G(x,y,z) == (x ∧ z) ∨(y ∧~ z)
- H(x,y,z) == x⊕y⊕ z
- I(x,y,z) == y⊕(x ∧ ~z)

Secure Hash Algorithm

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

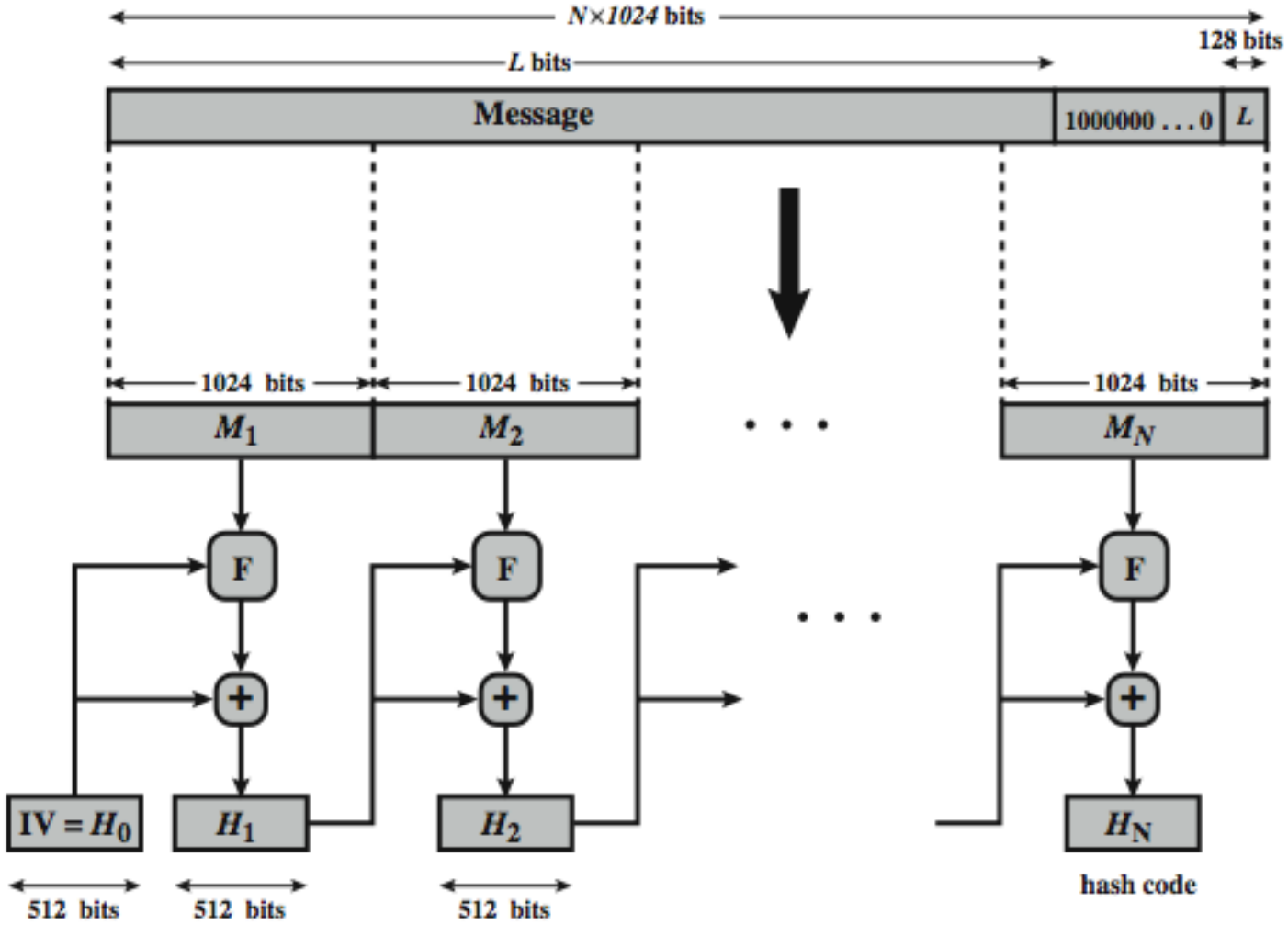
Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

SHA Versions

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message digest size	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	512	1024	1024
Word size	32	32	32	64	64
Number of steps	80	64	64	80	80

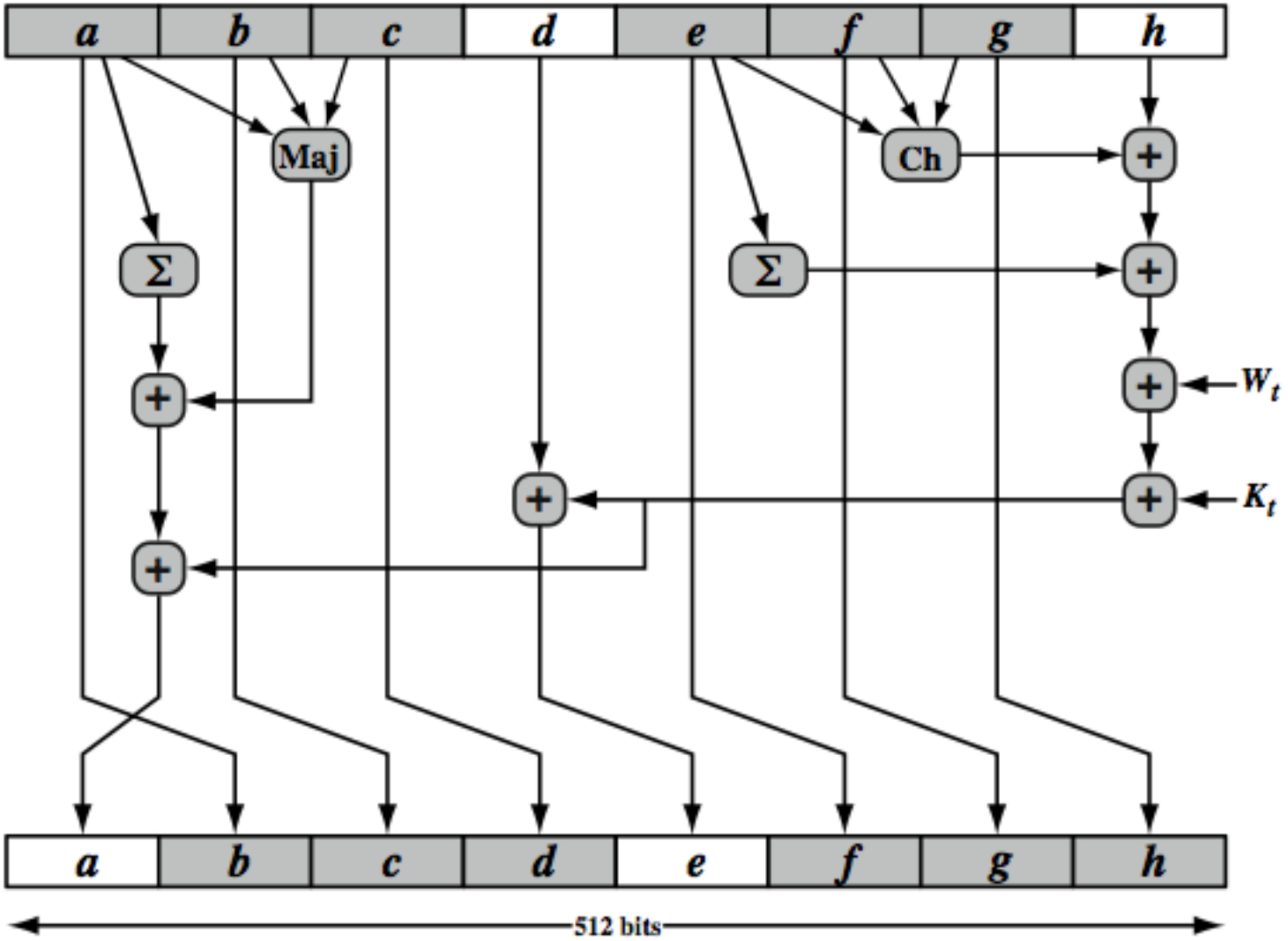
SHA-512 Overview



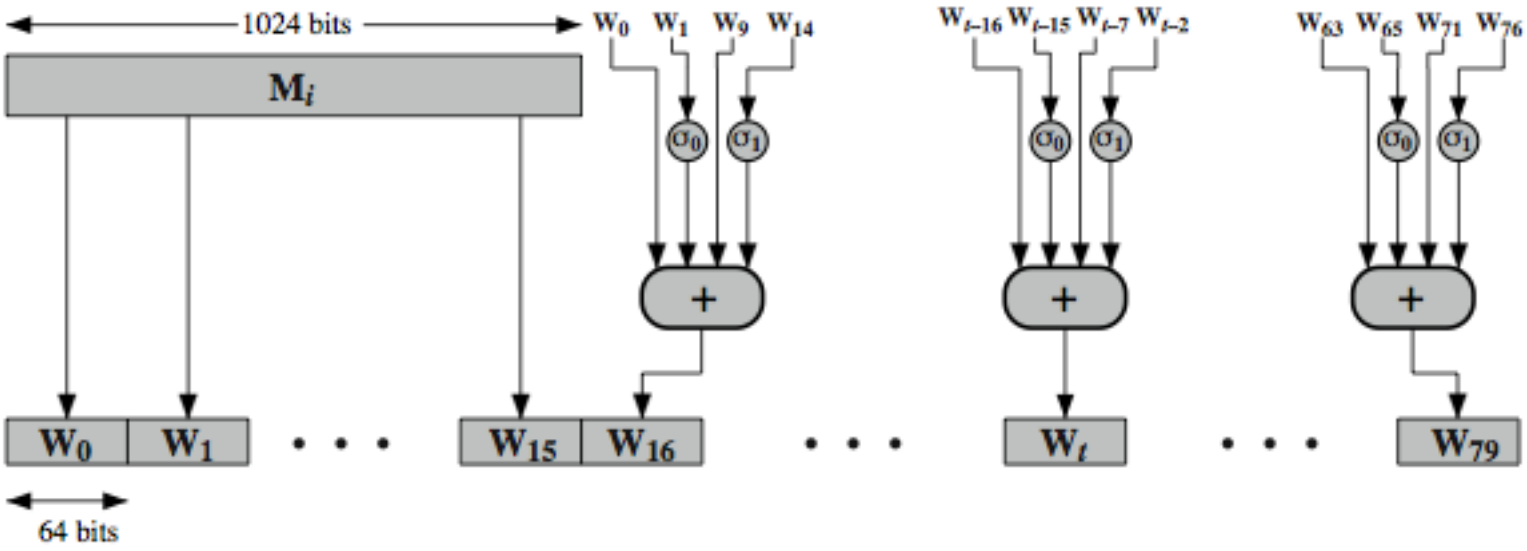
SHA-512 Compression Function

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
  - updating a 512-bit buffer
  - using a 64-bit value Wt derived from the current message block
  - and a round constant based on cube root of first 80 prime numbers

SHA-512 Round Function



SHA-512 Round Function



SHA-3

- SHA-1 not yet "broken"
  - but similar to broken MD5 & SHA-0
  - so considered insecure
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors so have concern
- NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function
  - goal to have in place by 2012 but not fixed

SHA-3 Requirements

- replace SHA-2 with SHA-3 in any use
  - so use same hash sizes
- preserve the online nature of SHA-2
  - so must process small blocks (512 / 1024 bits)
- evaluation criteria
  - security close to theoretical max for hash sizes
  - cost in time & memory
  - characteristics: such as flexibility & simplicity

Summary

- have considered:
  - hash functions
    - uses, requirements, security
  - hash functions based on block ciphers
  - SHA-1, SHA-2, SHA-3

Other Schemes

- MAC
- HMAC