Message Authentication

- message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- will consider the security requirements
- then three functions used in construction:
  - message encryption
  - message authentication code (MAC)
  - hash function

Security Requirements

- disclosure
- traffic analysis
- masquerade
- content modification
- sequence modification
- timing modification
- source repudiation
- destination repudiation

Message Encryption

- message encryption by itself also provides a measure of authentication
- if symmetric encryption is used then:
  - receiver know sender must have created it since only sender and receiver know key used
  - know content cannot have been altered if message has suitable structure, redundancy or a checksum to detect any changes

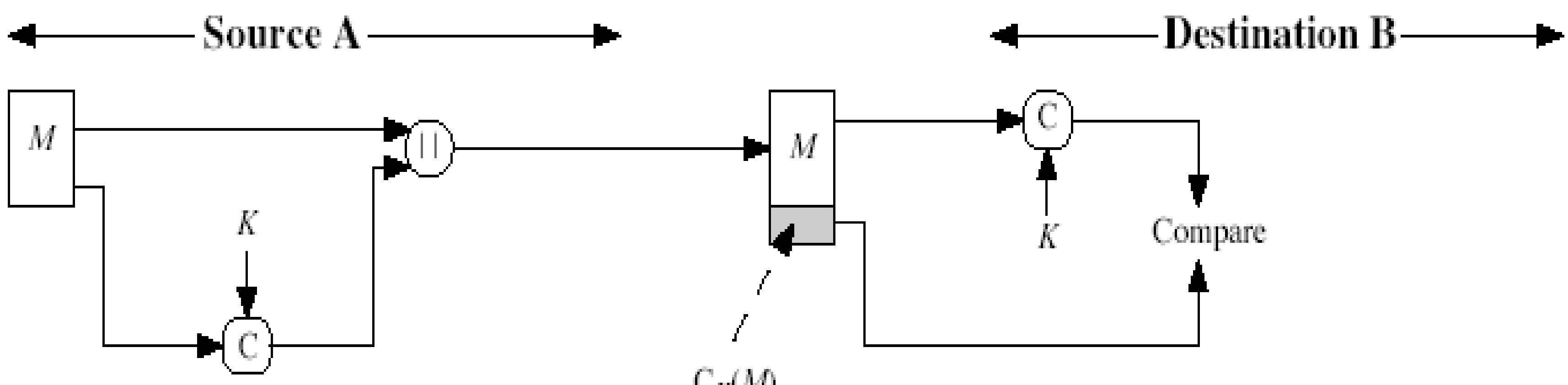
Message Encryption

- if public-key encryption is used:
  - encryption provides no confidence of sender
  - since anyone potentially knows public-key
  - however if
    - sender **signs** message using their private-key
    - then encrypts with recipients public key
    - have both secrecy and authentication
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message

Message Authentication Code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - like encryption though need not be reversible
- appended to message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

Message Authentication Code



(a) Message authentication

Message Authentication Codes

- as shown the MAC provides authenticity
- can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before
- why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (eg. archival use)
- note that a MAC is not a digital signature
  - what is the main difference?

MAC Properties

- a MAC is a cryptographic checksum

    $MAC = C_K(M)$

    - condenses a variable-length message M
    - using a secret key K
    - to a fixed-sized authenticator
- is a many-to-one function
    - potentially many messages have same MAC
    - but finding these needs to be very difficult

Requirements for MACs

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC (but not secret K), is infeasible to find another message with same MAC
  2. MAC should be uniformly distributed
  3. MAC should depend equally on all bits of the message

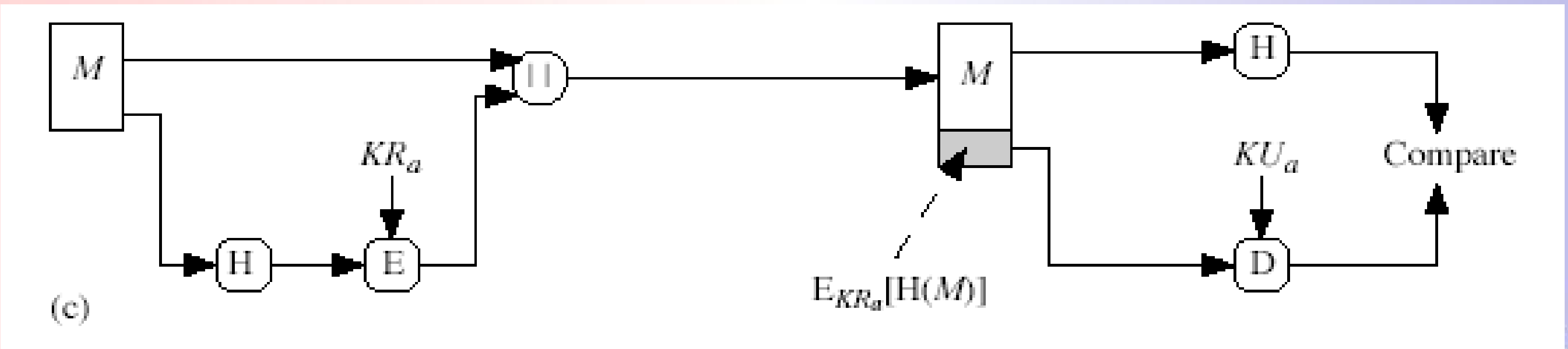
Using Symmetric Ciphers for MACs

- can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** is a widely used MAC based on DES-CBC
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits ($16 \le M \le 64$) of final block
- but final MAC is now too small for security

Hash Functions

- condenses arbitrary message to fixed size
- usually assume that the hash function is public and not keyed
  - cf. MAC which is keyed
- hash used to detect changes to message
- can use in various ways with message
- often to create a digital signature

Hash Functions & Digital Signatures



Hash Function Properties

- a Hash Function produces a fingerprint of some file/message/data

  `h = H(M)`

  - condenses a variable-length message M
  - to a fixed-sized fingerprint
- assumed to be public

Requirements for Hash Functions

1. can be applied to any sized message `M`
2. produces fixed-length output `h`
3. is easy to compute `h=H(M)` for any message `M`
4. given `h` is infeasible to find `x` s.t. `H(x)=h`
   - one-way property
5. given `x` is infeasible to find `y` s.t. `H(y)=H(x)`
   - weak collision resistance
6. is infeasible to find any pair `(x,y)` s.t. `H(y)=H(x)`
   - strong collision resistance

Simple Hash Functions

- are several proposals for simple functions
- based on XOR of message blocks
- not secure since can manipulate any message and either not change hash or change hash also
- need a stronger cryptographic function (next chapter)

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 not good
  - both due to birthday attack on 64-bit block
  - and "meet-in-the-middle" attack, which doesn't require multiple message generations
- other variants also susceptible to attack

Hash Functions & MAC Security

- like block ciphers have:
- **brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - have proposal for h/w MD5 cracker
    - 128-bit hash looks vulnerable, 160-bits better
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf key search) or MAC
    - at least 128-bit MAC is needed for security

Hash Functions & MAC Security

- **cryptanalytic attacks** exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
- have a number of analytic attacks on iterated hash functions
  - $CV_i = f[CV_{i-1}, M_i]$; $H(M)=CV_N$
  - typically focus on collisions in function f
  - like block ciphers is often composed of rounds
  - attacks exploit properties of round functions

Message Digest

- **Example 1**: Suppose we have 4000 and we divide it by 4 to get 1000.

Here 4 becomes fingerprint of 4000

If we are simply given 4 but further information is not given then we would not able to trace back the eq. 4 x 1000 = 4000

This is because there are infinite number of equations that gives result as 4.

- **Example 2:** Original Number is 7391743

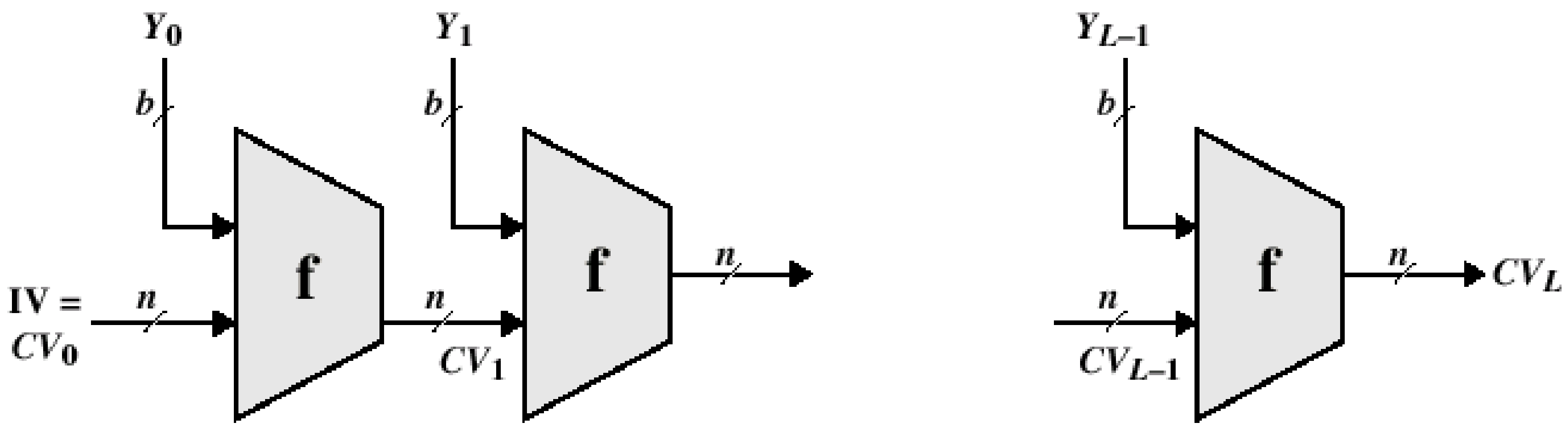
Operation	**Result**
Multiply 7 by 3	21
Discard first digit	1
Multiply 1 by 9	9
Multiply 9 by 1	9
Multiply 9 by 7	63
Discard first digit	3
Multiply 3 by 4	12
Discard first digit	2
Multiply 2 by 3	6

- **Message digest is 6**

MD5

- designed by Ronald Rivest (the R in RSA)
- latest in a series of MD2, MD4
- produces a 128-bit hash value
- until recently was the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns
- specified as Internet standard RFC1321

Hash Function Structure

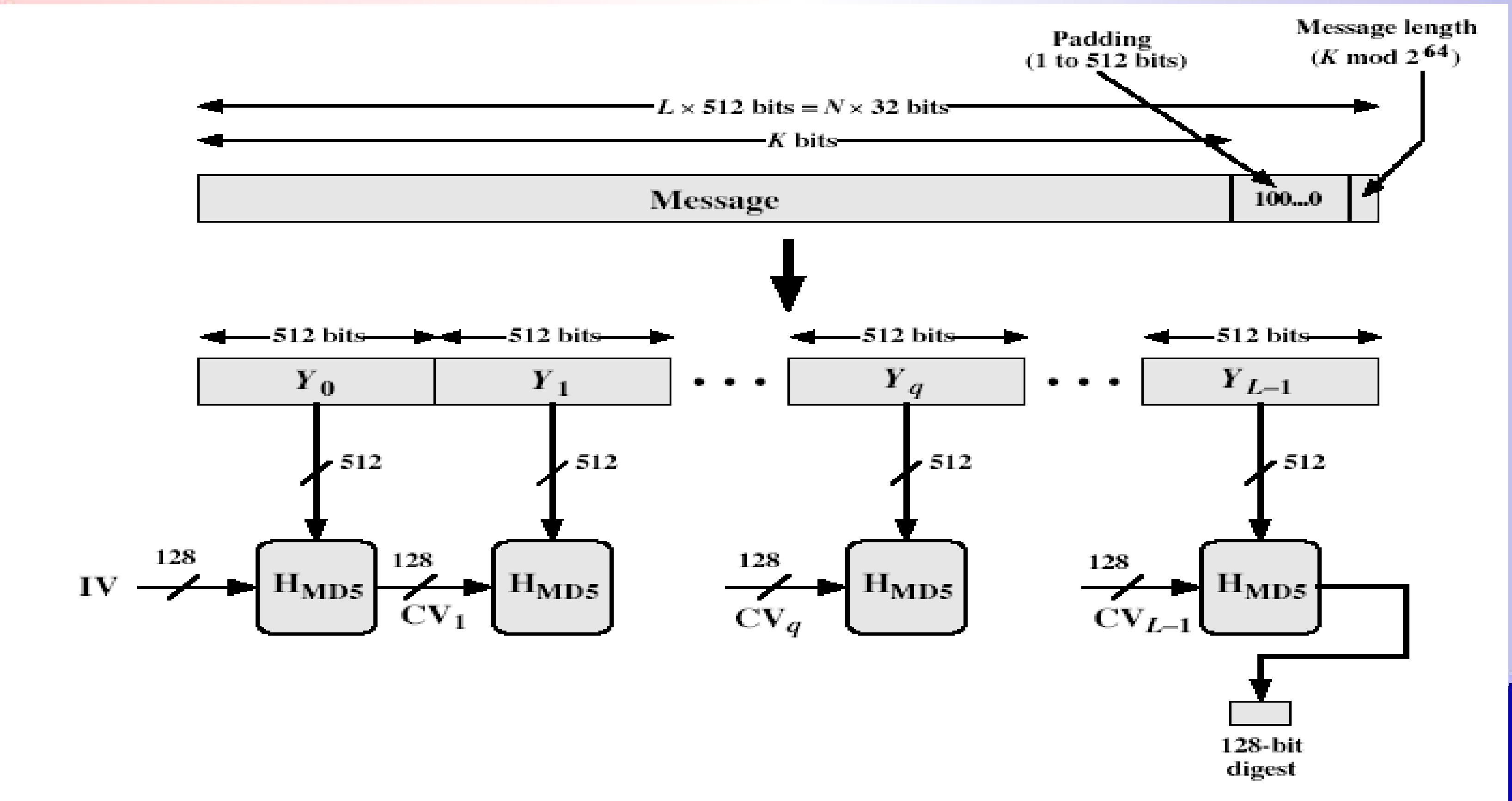


IV = Initial value
CV = chaining variable
Y_i = ith input block
f = compression algorithm
L = number of input blocks
n = length of hash code
b = length of input block

MD5 Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 4-word (128-bit) MD buffer (A,B,C,D)
4. process message in 16-word (512-bit) blocks:
   - using 4 rounds of 16 bit operations on message block & buffer
   - add output to buffer input to form new buffer value
5. output hash value is the final buffer value

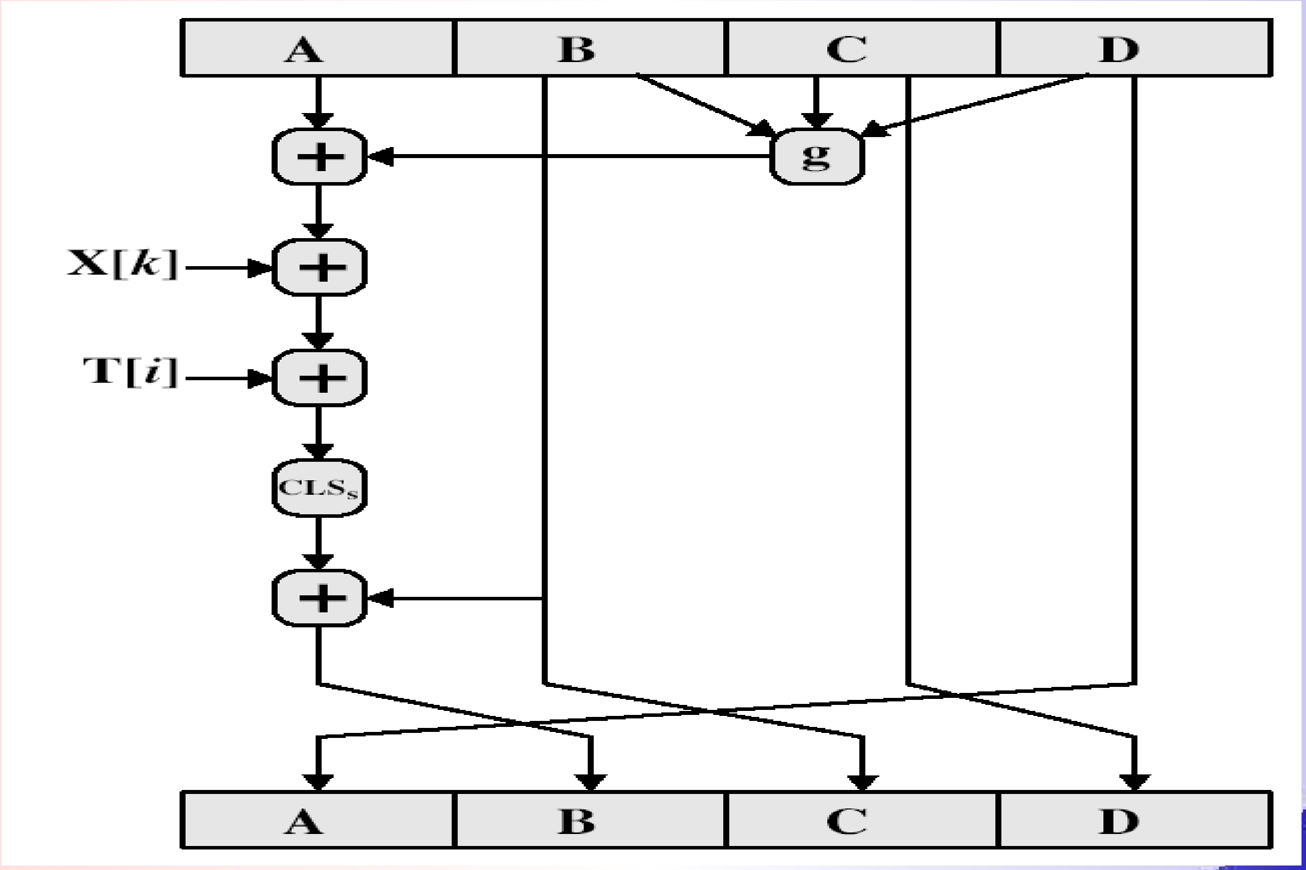
MD5 Overview



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)
- T[i] is a constant value derived from matrix of constants

MD5 Compression Function



MD4

- precursor to MD5
- also produces a 128-bit hash of message
- has 3 rounds of 16 steps vs 4 in MD5
- design goals:
  - collision resistant (hard to find collisions)
  - direct security (no dependence on "hard" problems)
  - fast, simple, compact
  - favours little-endian systems (eg PCs)

Strength of MD5

- MD5 hash is dependent on all message bits
- Strong security claims
- known attacks are:
  - Berson 92 attacked any 1 round using differential cryptanalysis (but can't extend)
  - Boer & Bosselaers 93 found a pseudo collision (again unable to extend)
  - Dobbertin 96 created collisions on MD compression function (but initial constants prevent exploit)
- conclusion is that MD5 looks vulnerable soon

Secure Hash Algorithm (SHA-1)

- SHA was designed by NIST & NSA in 1993, revised 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
- produces 160-bit hash values
- now the generally preferred hash algorithm
- based on design of MD4 with key differences

SHA Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 16-word (512-bit) chunks:
   - expand 16 words into 80 words by mixing & shifting
   - use 4 rounds of 20 bit operations on message block & buffer
   - add output to input to form new buffer value
5. output hash value is the final buffer value

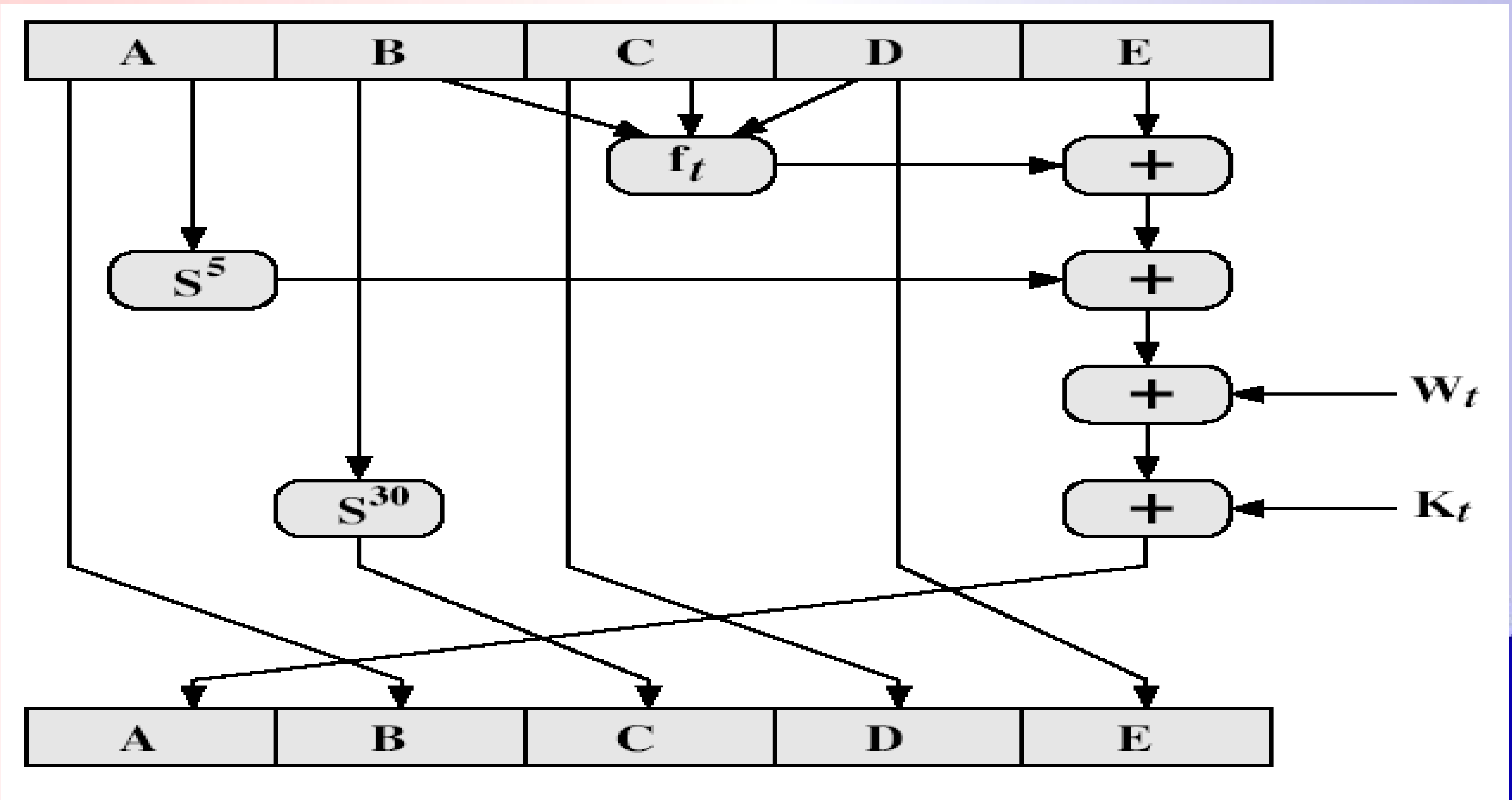
SHA-1 Compression Function

- each round has 20 steps which replaces the 5 buffer words thus:

  (A,B,C,D,E) <-
  (E+f(t,B,C,D)+(A<<5)+W_t+K_t),A,(B<<30),C,D)

- a,b,c,d refer to the 4 words of the buffer
- t is the step number
- f(t,B,C,D) is nonlinear function for round
- W_t is derived from the message block
- K_t is a constant value derived from sin

SHA-1 Compression Function



SHA-1 versus MD5

- brute force attack is harder (160 vs 128 bits for MD5)
- not vulnerable to any known attacks (compared to MD4/5)
- a little slower than MD5 (80 vs 64 steps)
- both designed as simple and compact
- optimised for big endian CPU's (vs MD5 which is optimised for little endian CPU’s)

Revised Secure Hash Standard

- NIST have issued a revision FIPS 180-2
- adds 3 additional hash algorithms
- 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

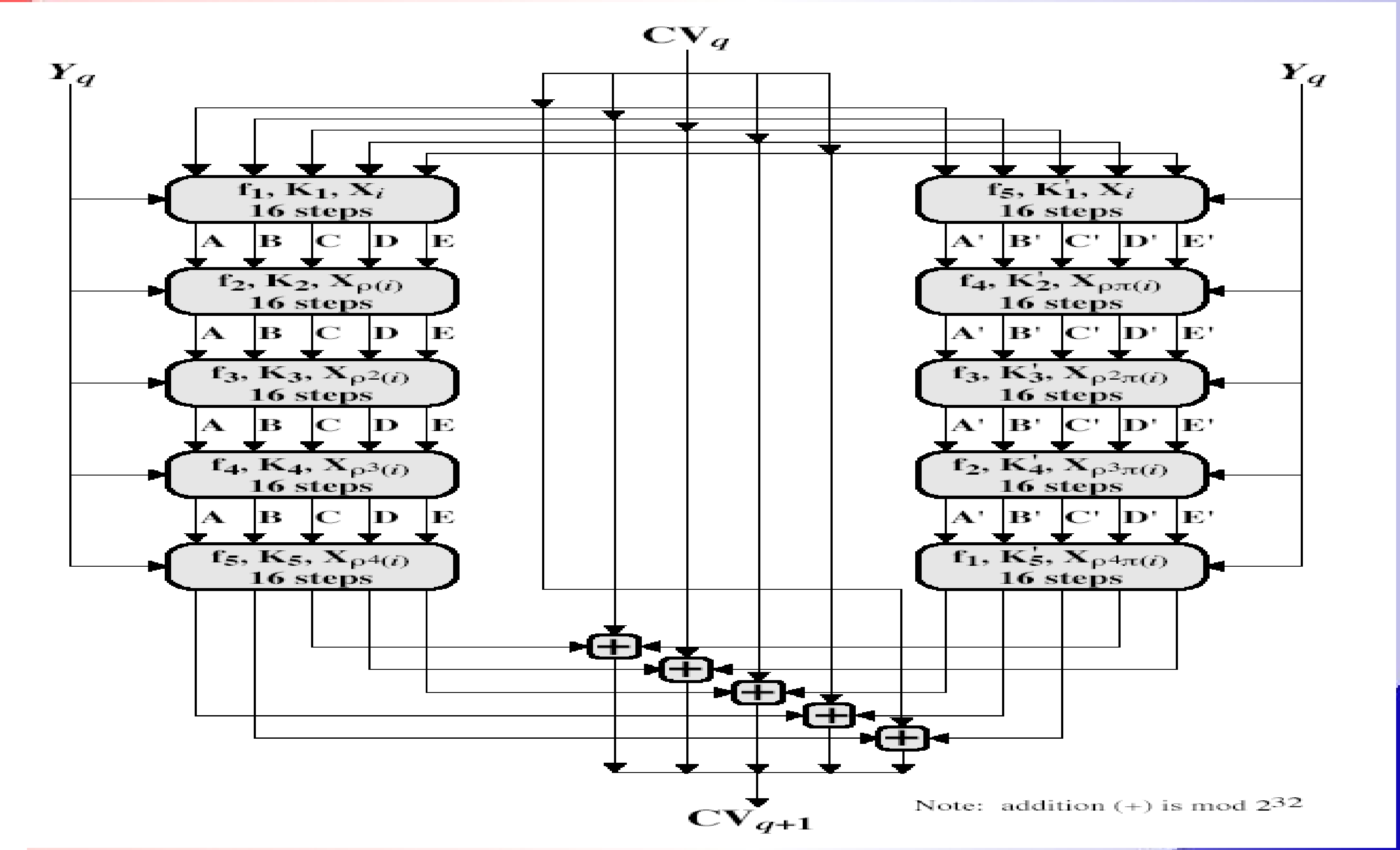
RIPEMD-160

- RIPEMD-160 was developed in Europe as part of RIPE project in 96
- by researchers involved in attacks on MD4/5
- initial proposal strengthen following analysis to become RIPEMD-160
- somewhat similar to MD5/SHA
- uses 2 parallel lines of 5 rounds of 16 steps
- creates a 160-bit hash value
- slower, but probably more secure, than SHA

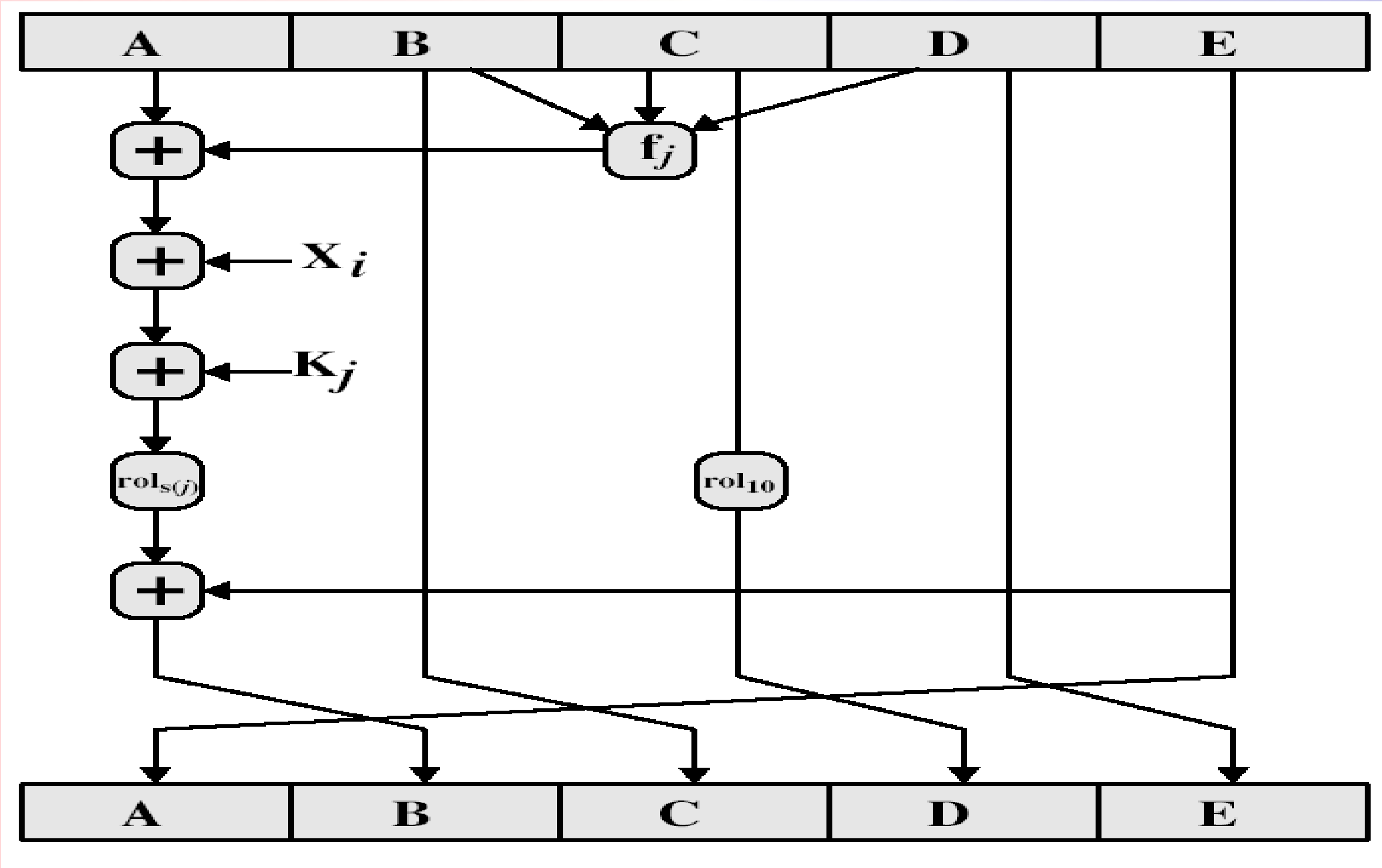
RIPEMD-160 Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 16-word (512-bit) chunks:
   - use 10 rounds of 16 bit operations on message block & buffer – in 2 parallel lines of 5
   - add output to input to form new buffer value
5. output hash value is the final buffer value

RIPEMD-160 Round



RIPEMD-160 Compression Function



RIPEMD-160 Design Criteria

- use 2 parallel lines of 5 rounds for increased complexity
- for simplicity the 2 lines are very similar
- step operation very close to MD5
- permutation varies parts of message used
- circular shifts designed for best results

RIPEMD-160 verses MD5 & SHA-1

- brute force attack harder (160 like SHA-1 vs 128 bits for MD5)
- not vulnerable to known attacks, like SHA-1 though stronger (compared to MD4/5)
- slower than MD5 (more steps)
- all designed as simple and compact
- SHA-1 optimised for big endian CPU's vs RIPEMD-160 & MD5 optimised for little endian CPU’s

Keyed Hash Functions as MACs

- have desire to create a MAC using a hash function rather than a block cipher
  - because hash functions are generally faster
  - not limited by export controls unlike block ciphers
- hash includes a key along with the message
- original proposal:

```
KeyedHash = Hash(Key|Message)
```

  - extension attack
- eventually led to development of HMAC

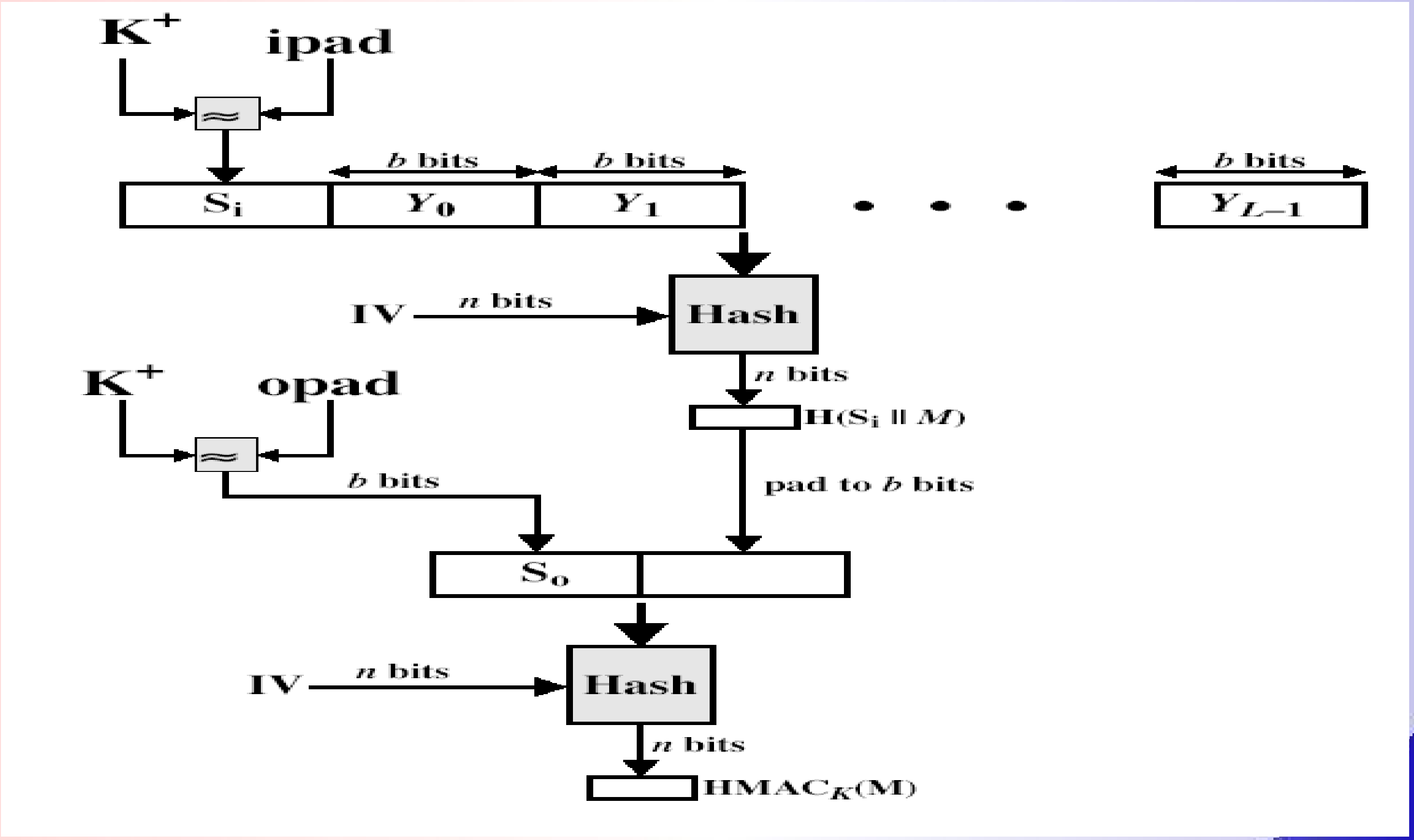
HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

$$HMAC_K = Hash[(K^+ \text{ XOR } opad) \,||\, Hash[(K^+ \text{ XOR } ipad)||M)]]$$

- where K^+ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash iterations than the message needs alone
- any of MD5, SHA-1, RIPEMD-160 can be used

HMAC Overview



HMAC Security

- know that the security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints

Summary

- have considered:
  - some current hash algorithms: MD5, SHA-1, RIPEMD-160
  - HMAC authentication using hash function