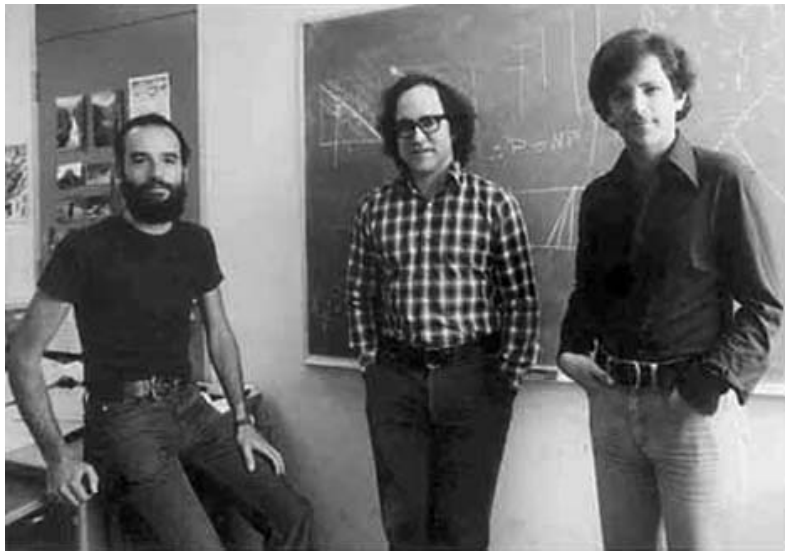
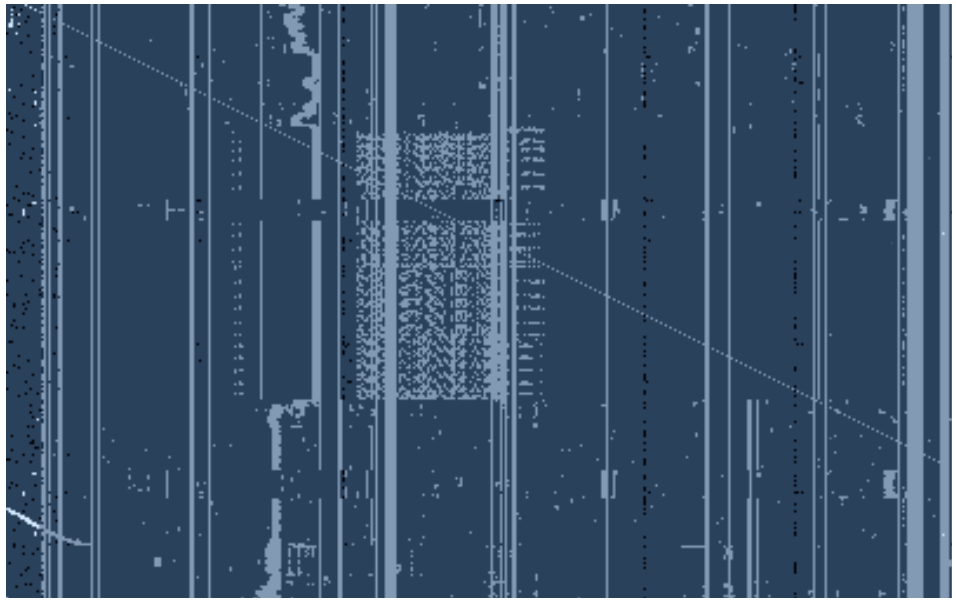




Crypto Basics

Micro-architectural attacks against AES

RSA

Message Digestion, Hashing, and MAC

Digital Signatures and Message Authentication

Micro-architectural attacks against AES

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - Exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random - p, q
- computing their system modulus $n=p.q$
  - note $\varnothing(n)=(p-1)(q-1)$
- selecting at random the encryption key e
    - where $1<e<\varnothing(n)$, $\gcd(e,\varnothing(n))=1$
- solve following equation to find decryption key d
  - $e.d=1 \bmod \varnothing(n)$ and $0 \le d \le n$
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d, $\varnothing$(n)}

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient $PU=\{e,n\}$
  - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key $PR=\{d\}$
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

Why RSA Works

- because of Euler's Theorem:
  - $a^{\phi(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\phi(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \phi(n)$
  - hence $e.d=1+k.\phi(n)$ for some k
- hence :

$$C^d = M^{e.d} = M^{1+k.\phi(n)} = M^1.(M^{\phi(n)})^k$$
$$= M^1.(1)^k = M^1 = M \bmod n$$

RSA Example - Key Setup

1. Select primes: $p=17$ & $q=11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de=1 \bmod 160$ and $d < 160$ Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key $PU=\{7,187\}$
7. Keep secret private key $PR=\{23,187\}$

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88<187$)
- encryption:

$$C = 88^7 \bmod 187 = 11$$

- decryption:

$$M = 11^{23} \bmod 187 = 88$$

Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4 . 7^1 = 3 . 7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128} . 3^1 = 5 . 3 = 4 \bmod 11$

Exponentiation

```
c = 0; f = 1
for i = k downto 0
    do c = 2 x c
       f = (f x f) mod n
    if b_i == 1 then
       c = c + 1
       f = (f x a) mod n
 return f
```

Efficient Encryption

- encryption uses exponentiation to power e
- hence if e small, this will be faster
  - often choose e=65537 (2^{16}-1)
  - also see choices of e=3 or e=17
- but if e too small (eg e=3) can attack
  - using Chinese remainder theorem & 3 messages with different modulii
- if e fixed must ensure `gcd(e,ø(n))=1`
  - ie reject any p or q not relatively prime to e

Efficient Decryption

- decryption uses exponentiation to power d
  - this is likely large, insecure if not
- can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique

RSA Key Generation

- users of RSA must:
  - determine two primes at random - `p, q`
  - select either `e` or `d` and compute the other
- primes `p,q` must not be easily derived from modulus `n=p.q`
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents `e, d` are inverses, so use Inverse algorithm to compute the other

RSA Security

- possible approaches to attacking RSA are:
  - brute force key search (infeasible given size of numbers)
  - mathematical attacks (based on difficulty of computing ø(n), by factoring modulus n)
  - timing attacks (on running of decryption)
  - chosen ciphertext attacks (given properties of RSA)

Factoring Problem

- mathematical approach takes 3 forms:
  - factor $n=p.q$, hence compute $\varnothing(n)$ and then d
  - determine $\varnothing(n)$ directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS
  - biggest improvement comes from improved algorithm
    - cf QS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

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 alternative functions used:
  - 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 now key used
  - know content cannot of been altered
  - if message has suitable structure, redundancy or a checksum to detect any changes

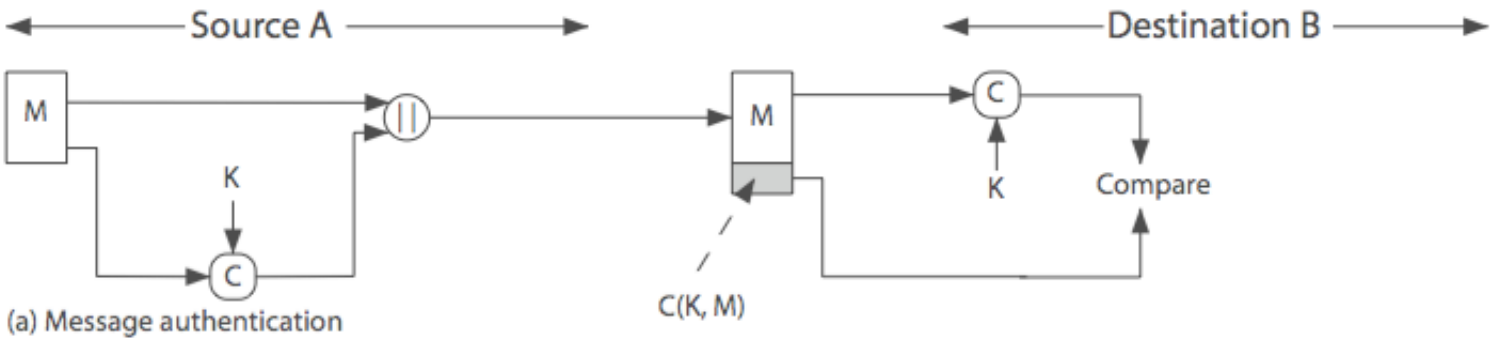
Message Encryption

- 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 authentication
- 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

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

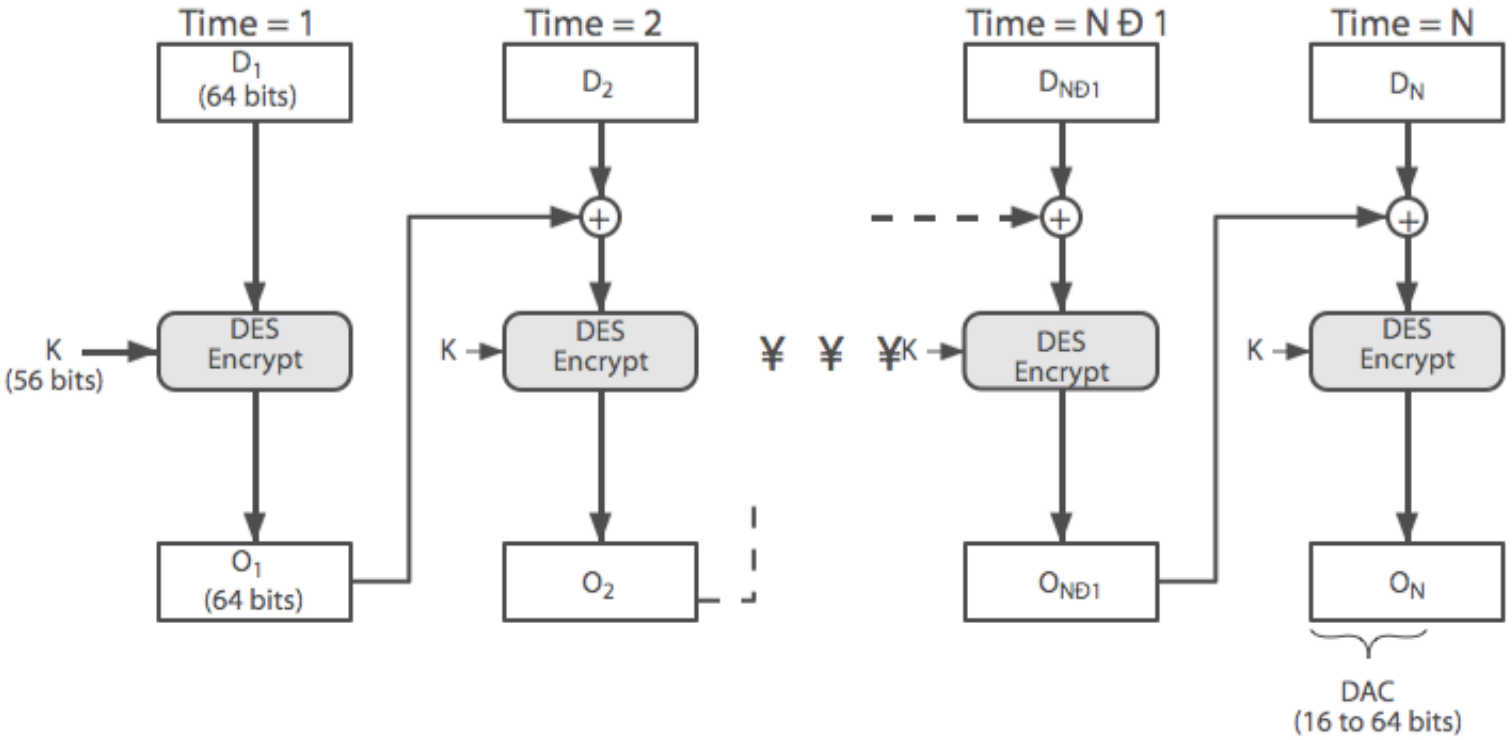
Requirements for MACs

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, is infeasible to find another message with same MAC
  2. MACs 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≤M≤64) of final block
- but final MAC is now too small for security

Data Authentication Algorithm



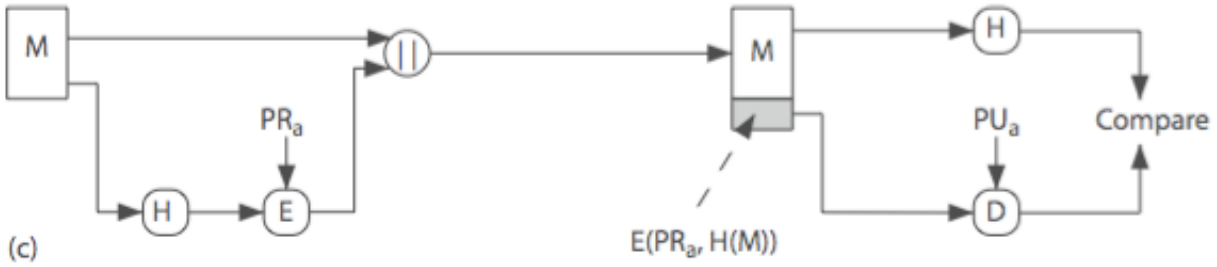
Hash Functions

- condenses arbitrary message to fixed size

  `h = H(M)`

- 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
- most often to create a digital signature

Hash Functions & Digital Signatures



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 `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

Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - opponent generates $2^{m/2}$ variations of a valid message all with essentially the same meaning
  - opponent also generates $2^{m/2}$ variations of a desired fraudulent message
  - 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

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

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

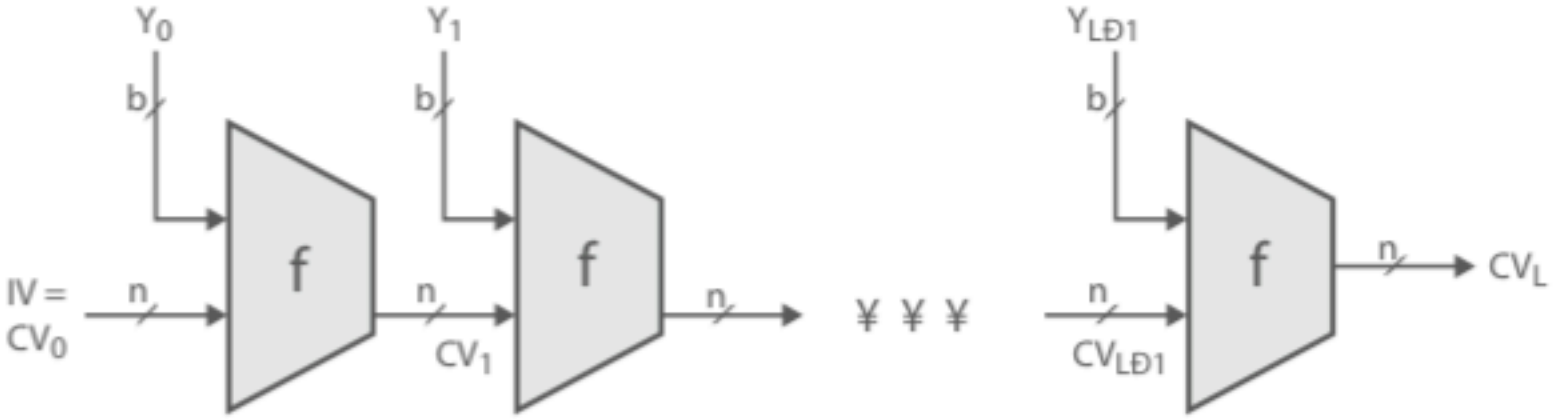
Summary

- have considered:
  - message authentication using
  - message encryption
  - MACs
  - hash functions
  - general approach & security

Hash and MAC Algorithms

- Hash Functions
  - condense arbitrary size message to fixed size
  - by processing message in blocks
  - through some compression function
  - either custom or block cipher based
- Message Authentication Code (MAC)
  - fixed sized authenticator for some message
  - to provide authentication for message
  - by using block cipher mode or hash function

Hash Algorithm Structure



IV = Initial value
CV_i = 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

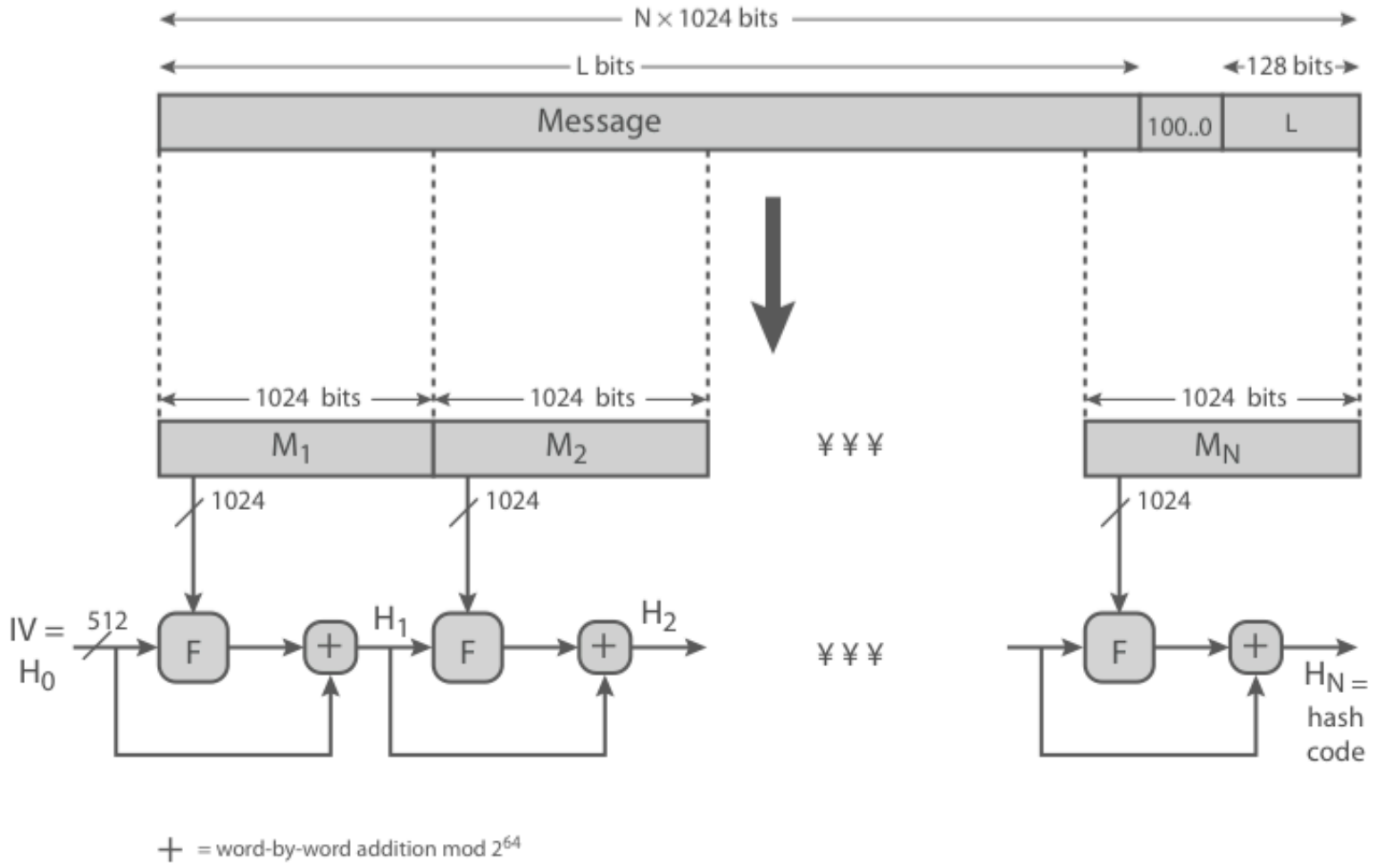
Secure Hash Algorithm

- 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

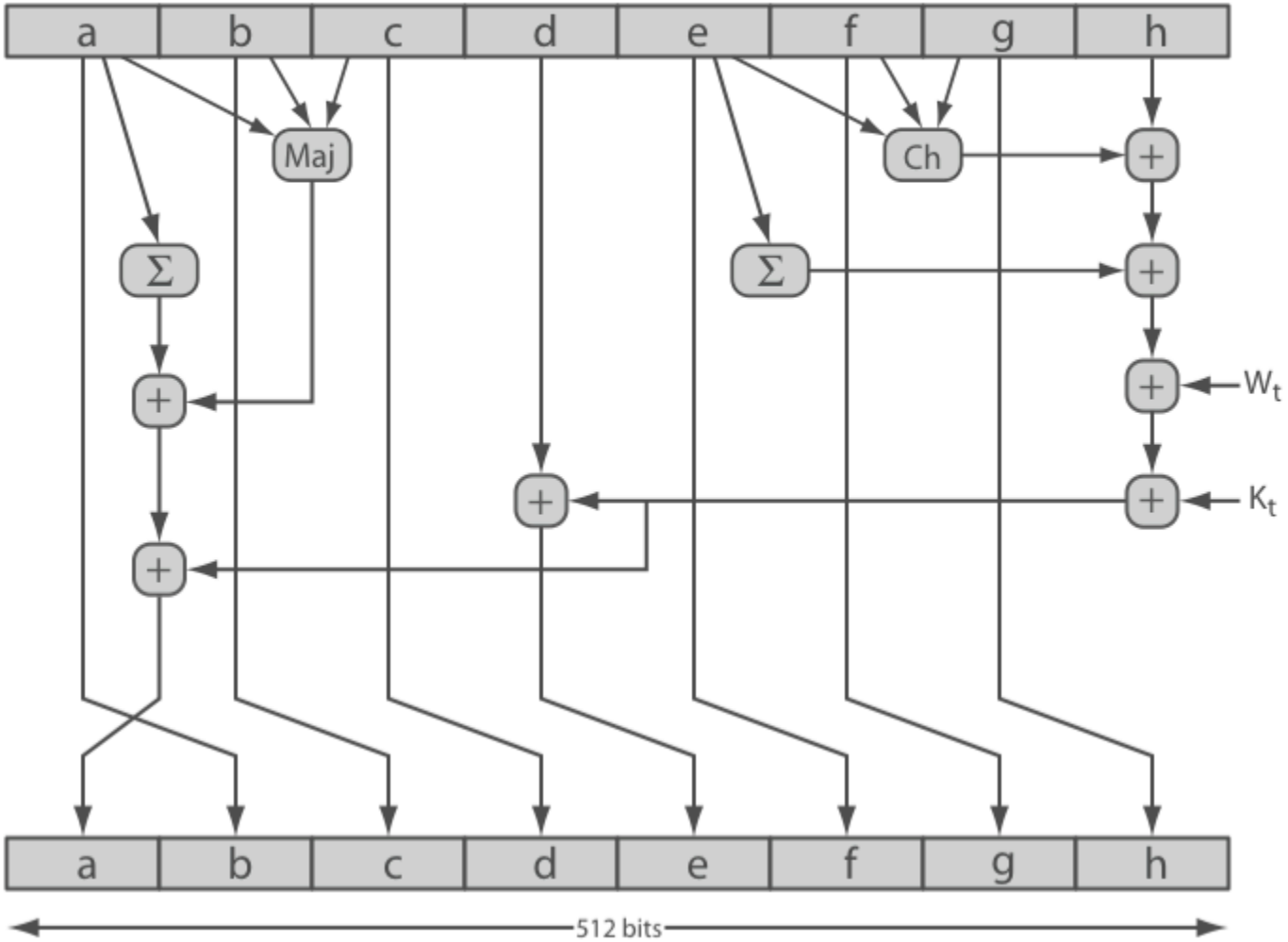
SHA-512 Overview



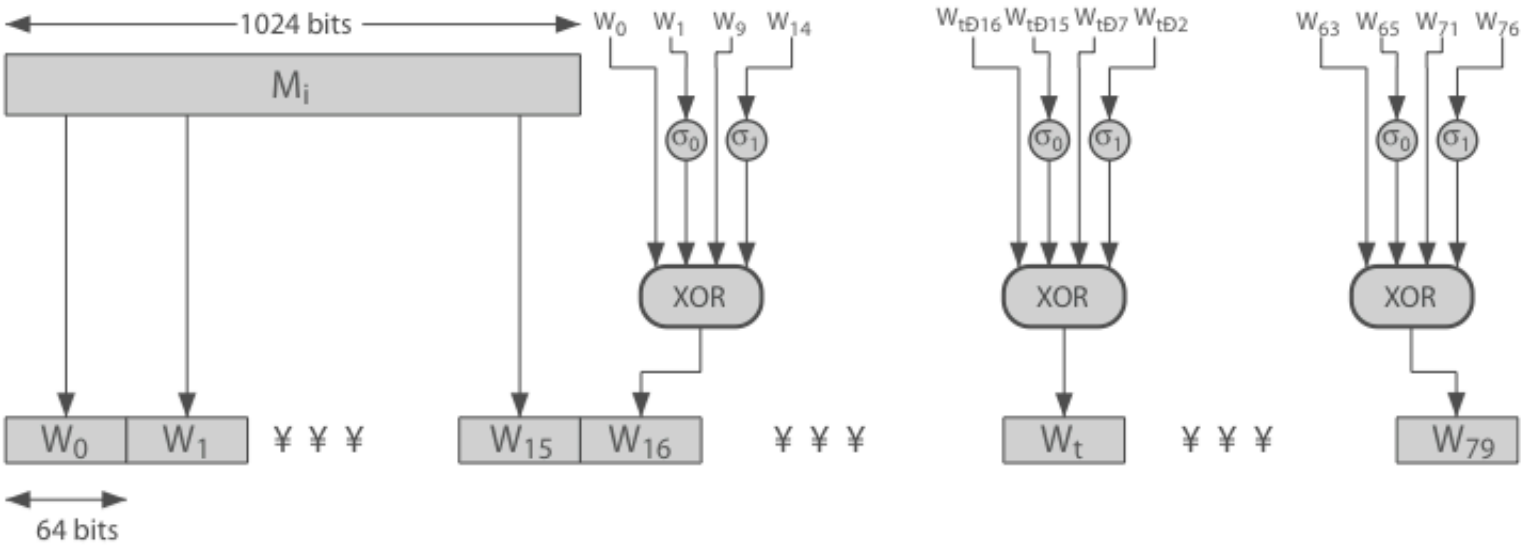
SHA-512 Compression Function

- 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



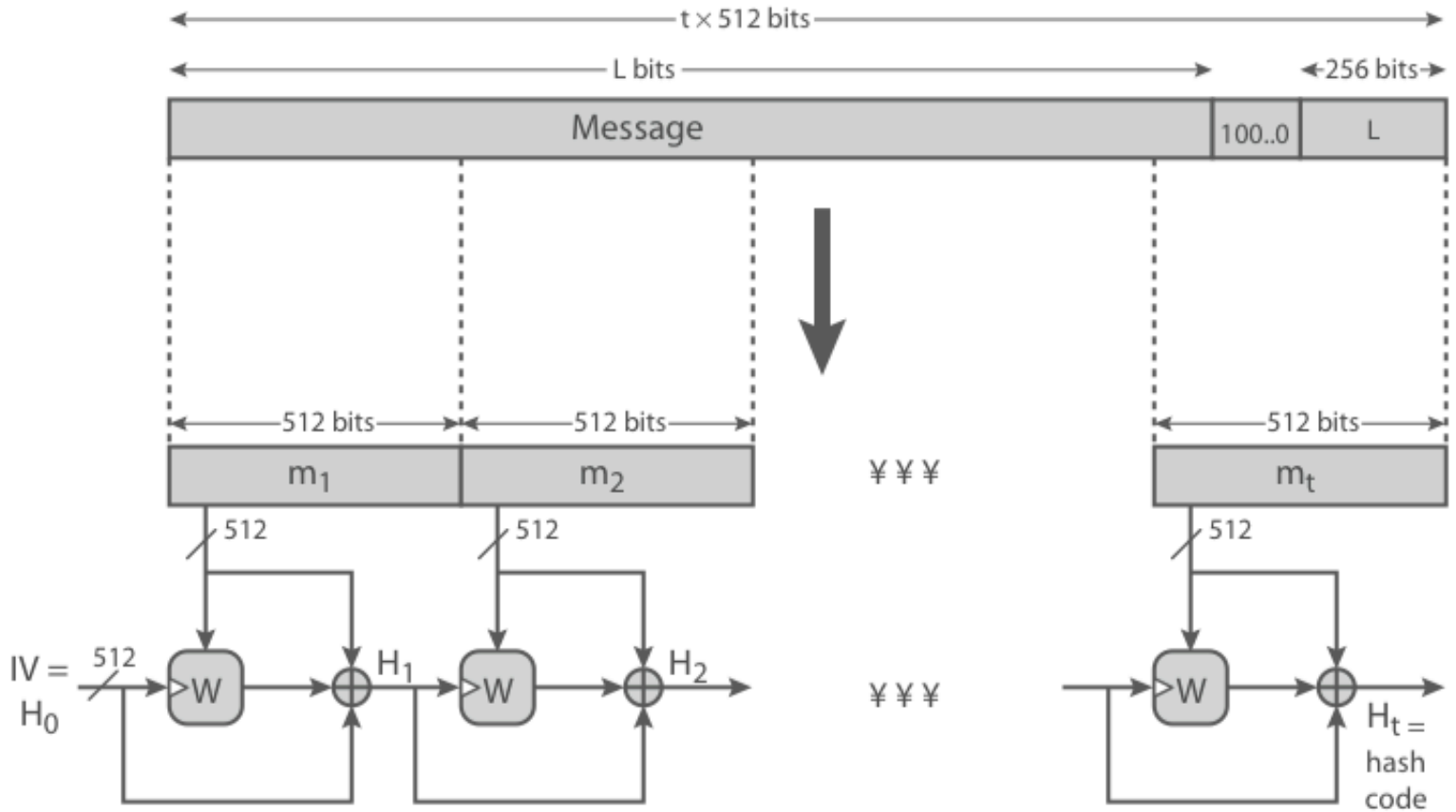
SHA-512 Round Function



Whirlpool

- now examine the Whirlpool hash function
- endorsed by European NESSIE project
- uses modified AES internals as compression function
- addressing concerns on use of block ciphers seen previously
- with performance comparable to dedicated algorithms like SHA

Whirlpool Overview

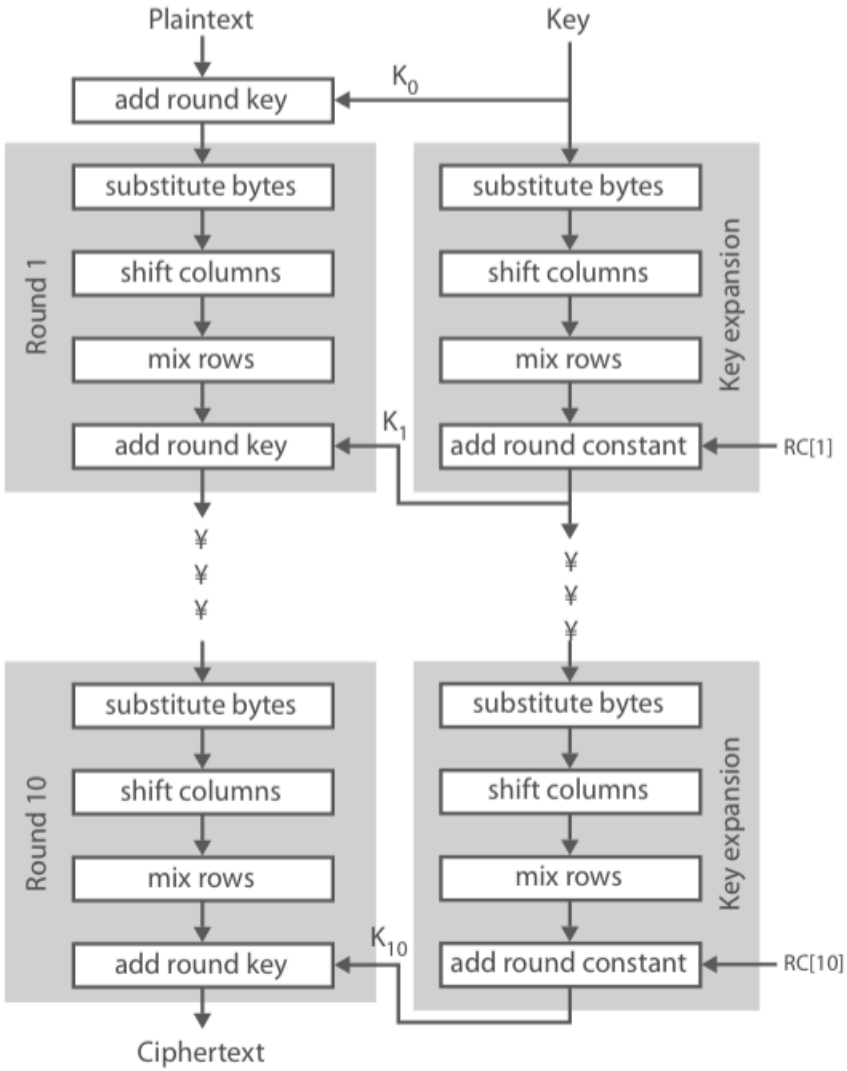


Note: triangular hatch marks key input

Whirlpool Block Cipher W

- designed specifically for hash function use
- with security and efficiency of AES
- but with 512-bit block size and hence hash
- similar structure & functions as AES but
  - input is mapped row wise
  - has 10 rounds
  - a different primitive polynomial for GF(2^8)
  - uses different S-box design & values

Whirlpool Block Cipher W



Whirlpool Performance & Security

- Whirlpool is a very new proposal
- hence little experience with use
- but many AES findings should apply
- does seem to need more h/w than SHA, but with better resulting performance

Keyed Hash Functions as MACs

- want a MAC based on a hash function
  - because hash functions are generally faster
  - code for crypto hash functions widely available
- hash includes a key along with message
- original proposal:

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

  - some weaknesses were found with this
- eventually led to development of HMAC

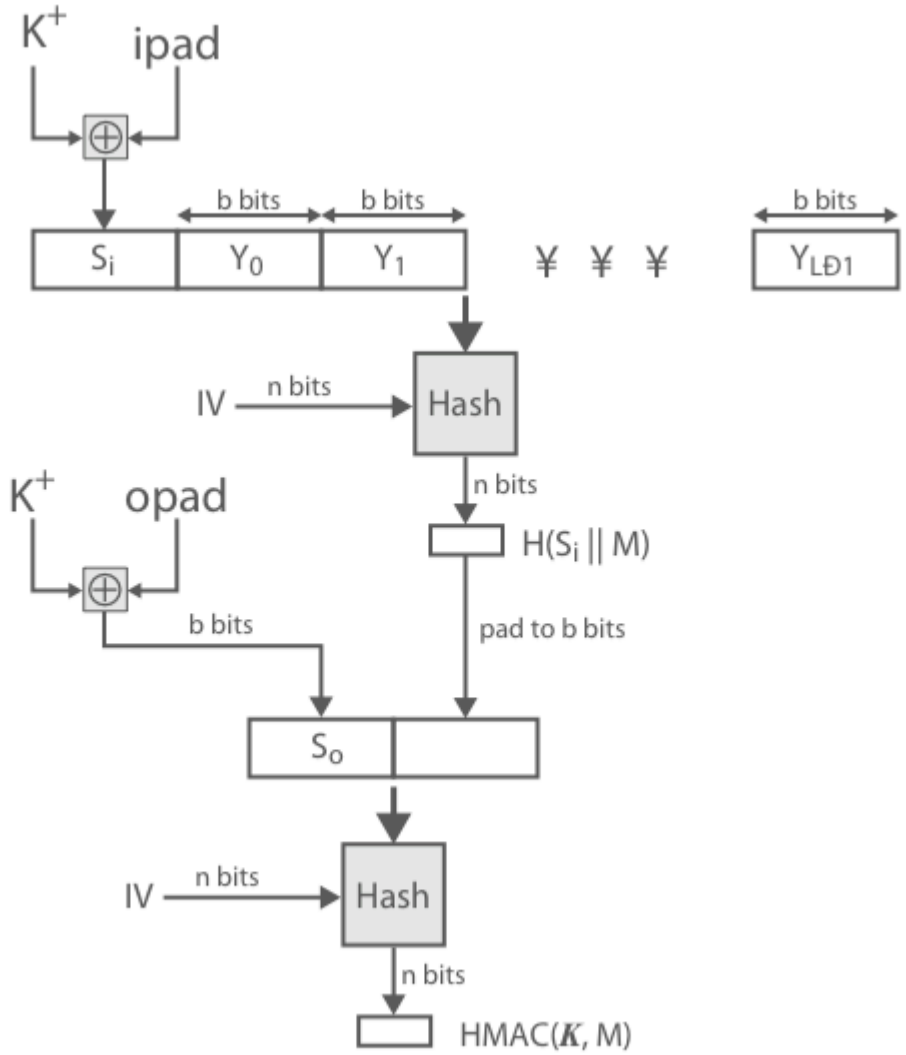
HMAC

- 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 calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Overview



HMAC Security

- proved 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
    - SHA-512 & Whirlpool
  - HMAC authentication using hash function

Digital Signatures

- have looked at message authentication
  - but does not address issues of lack of trust
- digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- hence include authentication function with additional capabilities

Digital Signature Properties

- must depend on the message signed
- must use information unique to sender
  - to prevent both forgery and denial
- must be relatively easy to produce
- must be relatively easy to recognize & verify
- be computationally infeasible to forge
  - with new message for existing digital signature
  - with fraudulent digital signature for given message
- be practical save digital signature in storage

Direct Digital Signatures

- involve only sender & receiver
- assumed receiver has sender's public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using receivers public-key
- important that sign first then encrypt message & signature
- security depends on sender's private-key

Arbitrated Digital Signatures

- involves use of arbiter A
  - validates any signed message
  - then dated and sent to recipient
- requires suitable level of trust in arbiter
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message

Authentication Protocols

- used to convince parties of each others identity and to exchange session keys
- may be one-way or mutual
- key issues are
  - confidentiality – to protect session keys
  - timeliness – to prevent replay attacks
- published protocols are often found to have flaws and need to be modified

Replay Attacks

- where a valid signed message is copied and later resent
  - simple replay
  - repetition that can be logged
  - repetition that cannot be detected
  - backward replay without modification
- countermeasures include
  - use of sequence numbers (generally impractical)
  - timestamps (needs synchronized clocks)
  - challenge/response (using unique nonce)

Using Public-Key Encryption

- have a range of approaches based on the use of public-key encryption
- need to ensure have correct public keys for other parties
- using a central Authentication Server (AS)
- various protocols exist using timestamps or nonces

One-Way Authentication

- required when sender & receiver are not in communications at same time (eg. email)
- have header in clear so can be delivered by email system
- may want contents of body protected & sender authenticated

Using Symmetric Encryption

- can refine use of KDC but can't have final exchange of nonces, vis:
  1. A->KDC: ID_A || ID_B || N_1
  2. KDC -> A: E_{Ka}[Ks || ID_B || N_1 || E_{Kb}[Ks||ID_A]]
  3. A -> B: E_{Kb}[Ks||ID_A] || E_{Ks}[M]
- does not protect against replays
  - could rely on timestamp in message, though email delays make this problematic

Public-Key Approaches

- have seen some public-key approaches
- if confidentiality is major concern, can use:

  A->B: $E_{PUb}[Ks] \,||\, E_{Ks}[M]$

  - has encrypted session key, encrypted message
- if authentication needed use a digital signature with a digital certificate:

  A->B: $M \,||\, E_{PRa}[H(M)] \,||\, E_{PRas}[T||ID_A||PU_a]$

  - with message, signature, certificate

Digital Signature Standard (DSS)

- US Govt approved signature scheme
- designed by NIST & NSA in early 90's
- published as FIPS-186 in 1991
- revised in 1993, 1996 & then 2000
- uses the SHA hash algorithm
- DSS is the standard, DSA is the algorithm
- FIPS 186-2 (2000) includes alternative RSA & elliptic curve signature variants