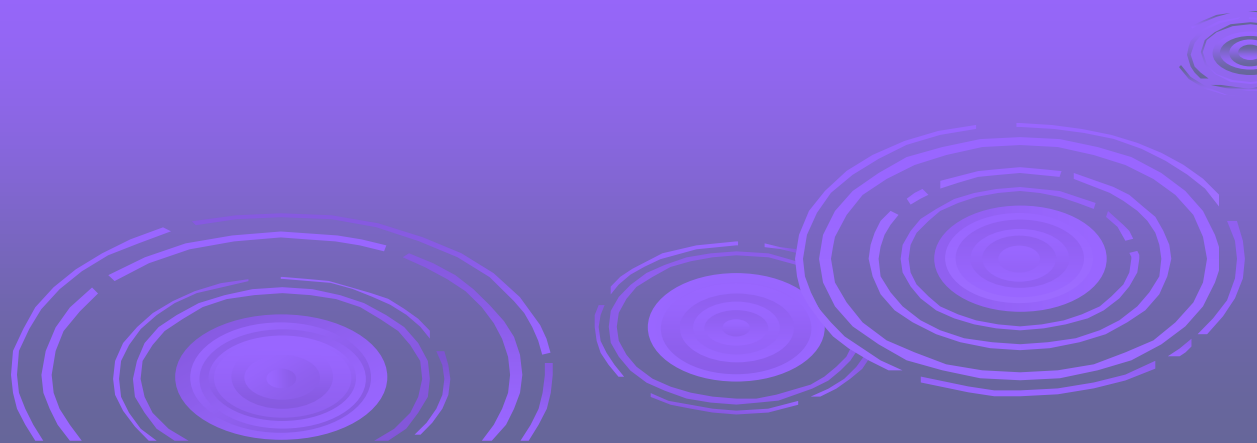
Cryptography and Network Security Chapter 11



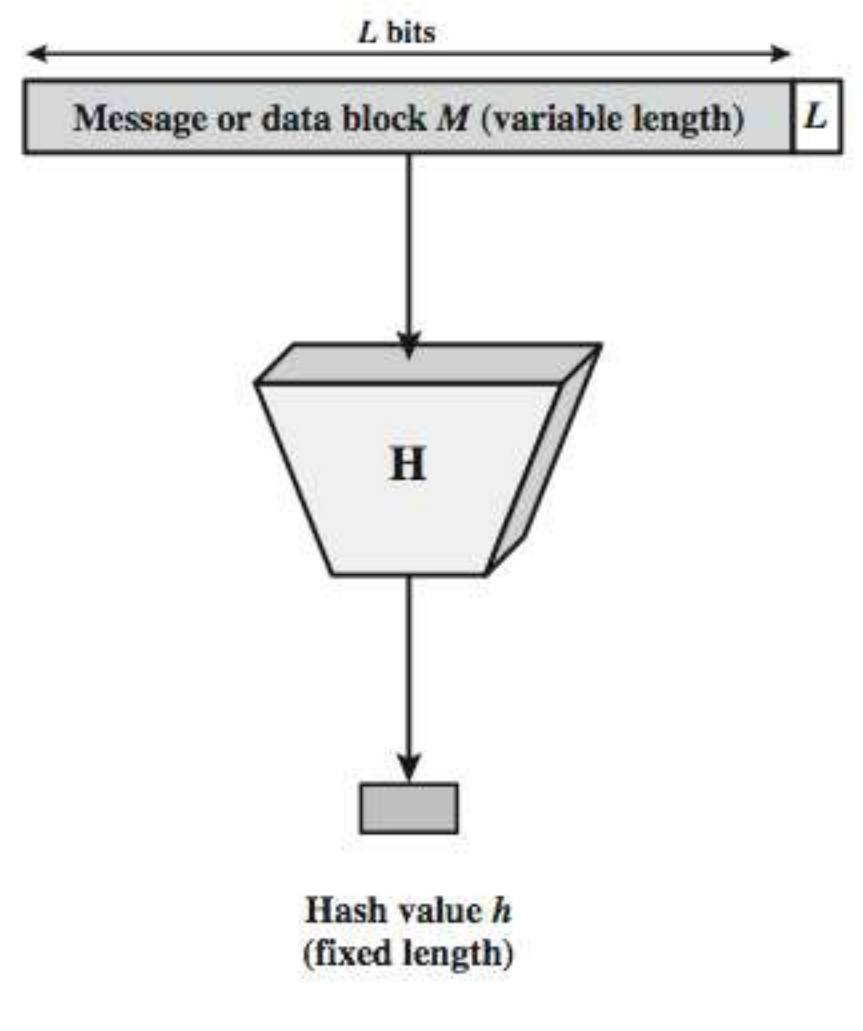
Hash Functions

- condenses arbitrary message to fixed size

  `h = H(M)`

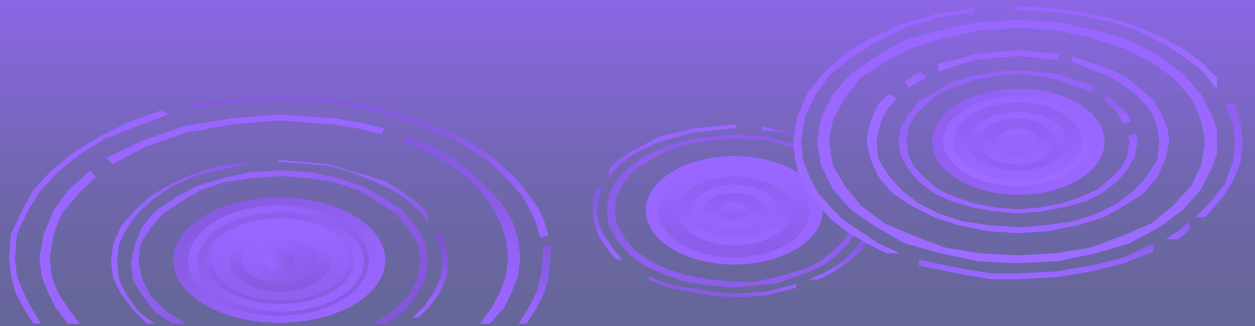
- usually assume hash function is public
- hash used to detect changes to message
- want a cryptographic hash function
  - computationally infeasible to find data mapping to specific hash (one-way property)
  - computationally infeasible to find two data to same hash (collision-free property)

Cryptographic Hash Function



Hash Function Uses

- Message Authentication Code (MAC)
  - send keyed hash of message
  - MAC, message optionally encrypted
- Digital Signature (non-repudiation)
  - Encrypt hash with private (signing) key
  - Verify with public (verification) key
- Other uses

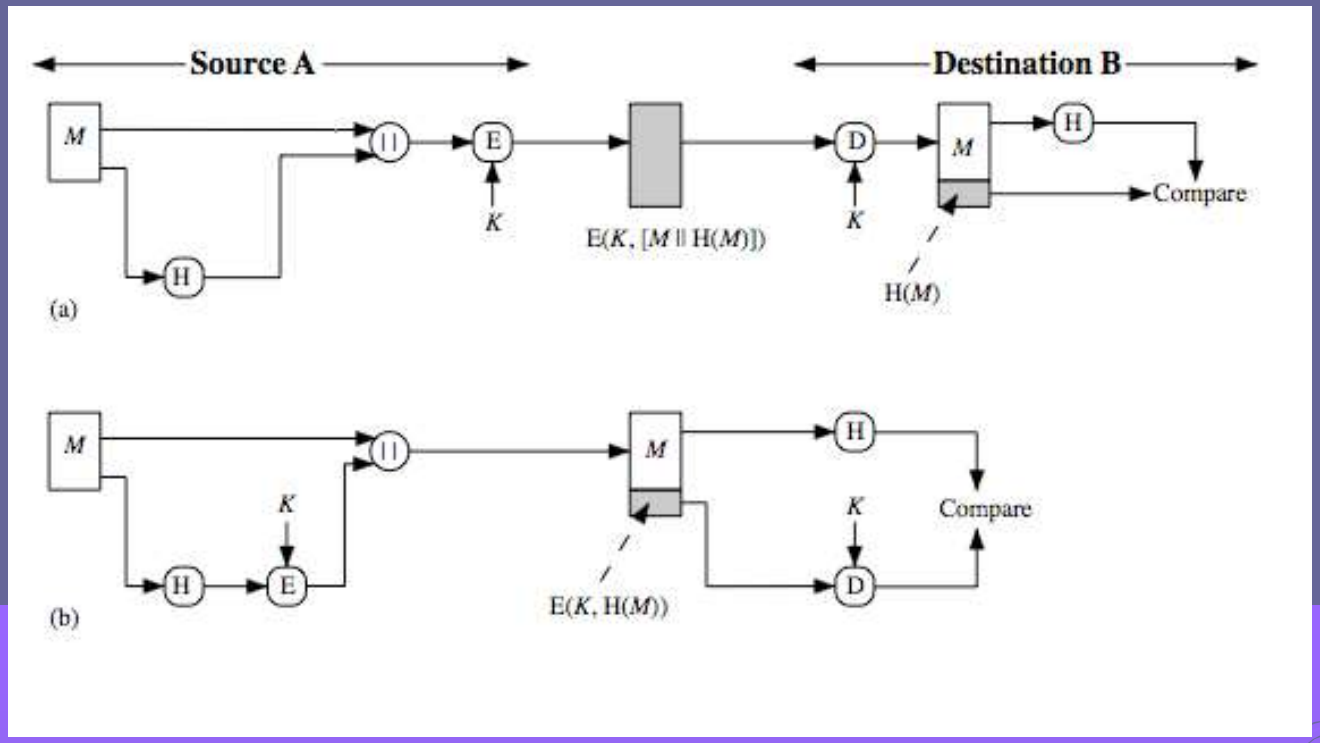


Hash Functions & Message Authentication

Symmetric Key Unkeyed Hash

a) Message encrypted

b) Message unencrypted

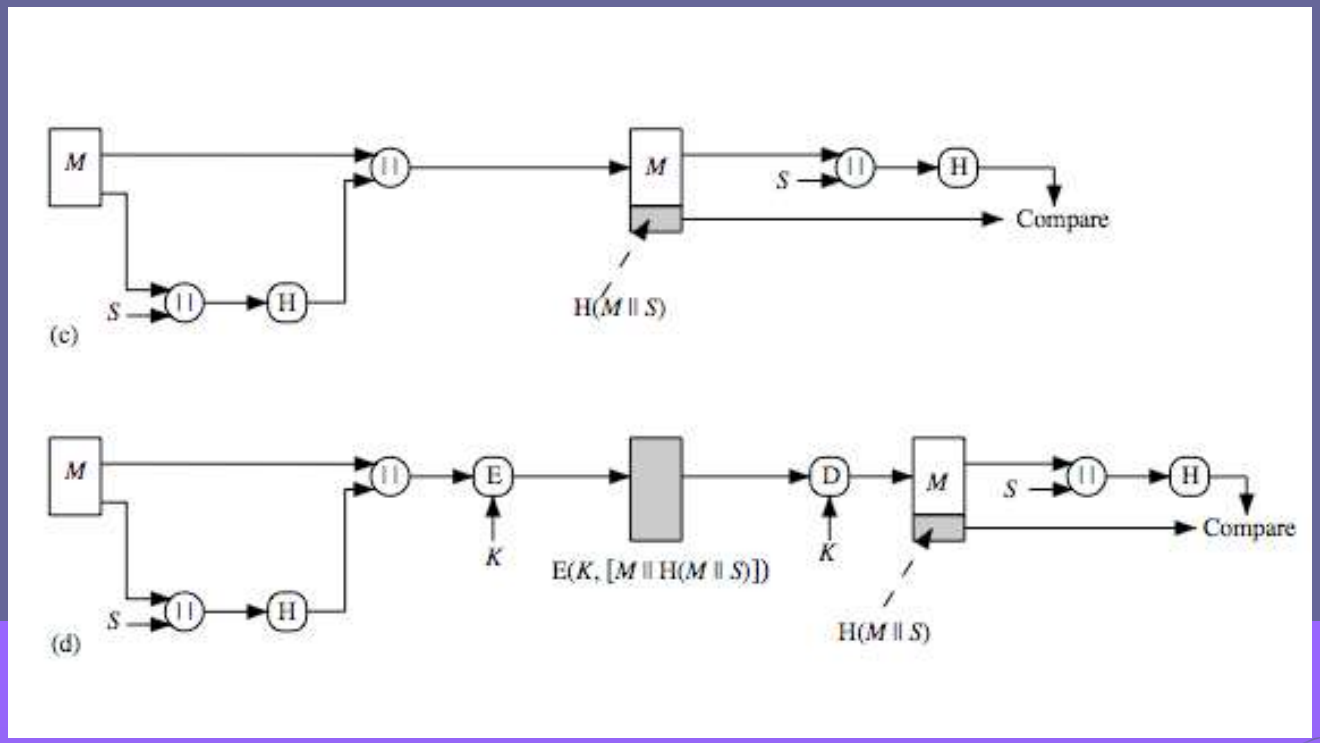


Hash Functions & Message Authentication

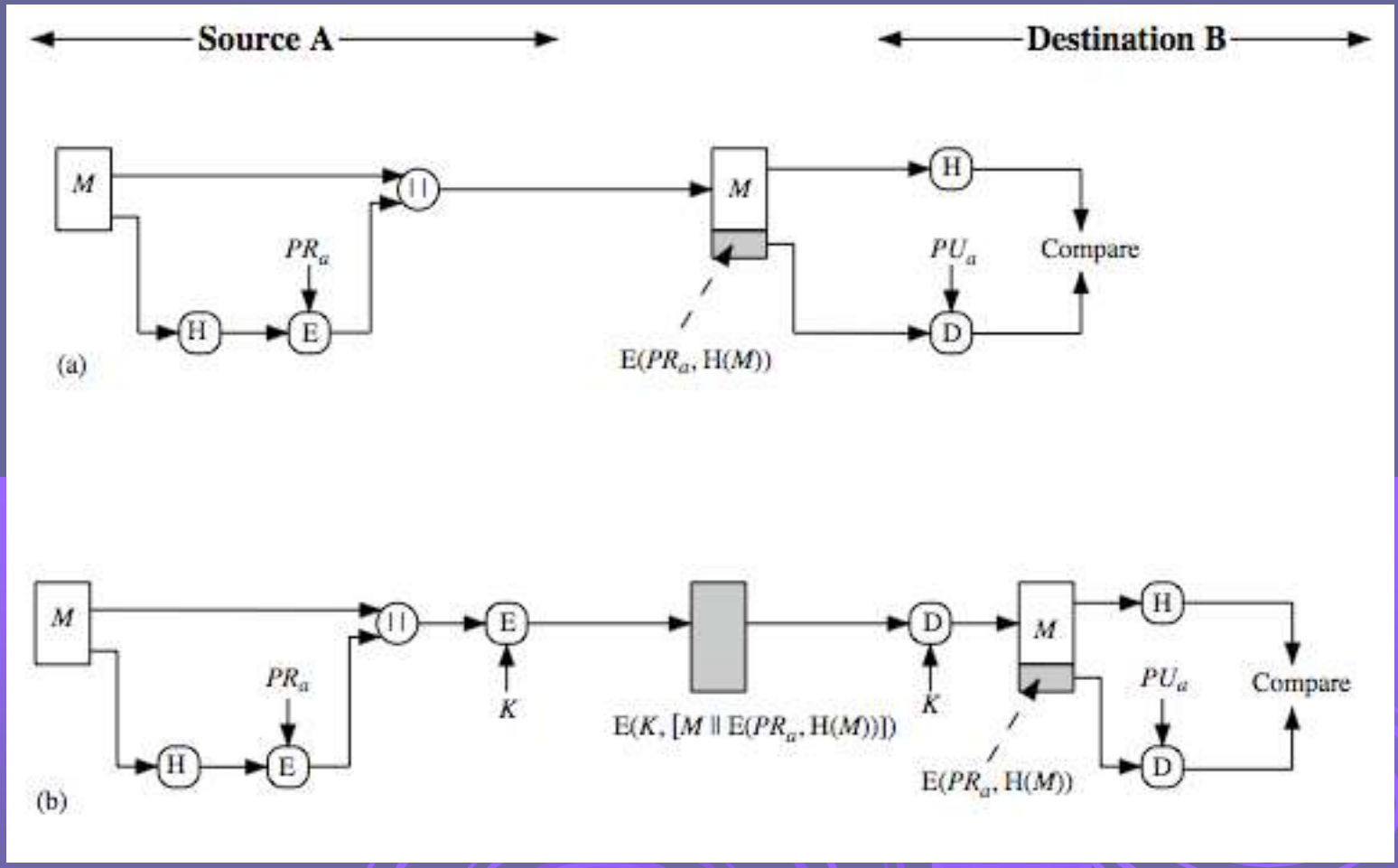
Symmetric Key Keyed Hash

a) Message unencrypted

d) Message encrypted



Hash Functions & Digital Signatures

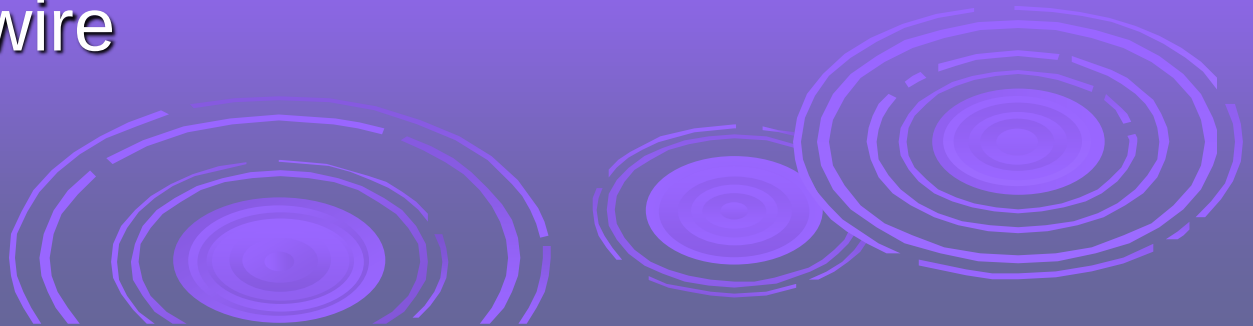


Other Hash Function Uses

- pseudorandom function (PRF)
  - Generate session keys, nonces
  - Produce key from password
  - Derive keys from master key cooperatively
- pseudorandom number generator (PRNG)
  - Vernam Cipher/OTP
  - S/Key, proof of “what you have” via messages

More Hash Function Uses

- to create a one-way password file
  - store hash of password not actual password
  - e.g., Unix, Windows NT, etc.
  - salt to deter precomputation attacks
  - Rainbow tables
- for intrusion detection and virus detection
  - keep & check hash of files on system
  - e.g., Tripwire



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 $y \neq x$ 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

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

Block Ciphers as Hash Functions

Block cipher key length B

Pad Message M to multiple of B

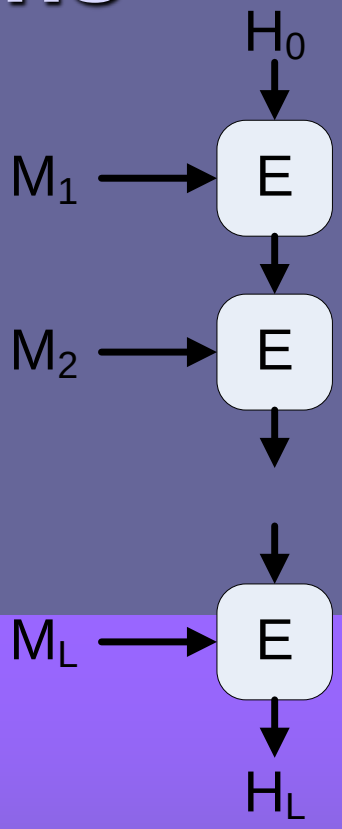
Break padded M into L blocks

$L = |M|/B$

$M = M_1\ M_2\ \ldots\ M_L$

Use blocks of M as keys in block cipher, iteratively encrypt state value starting with constant H_0 resulting in hash value

$H = H_L = E(M_L,\ldots.E(M_2,E(M_1,H_0))\ldots)$



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
- 2005 results on security of SHA-1 raised concerns on its use in future applications

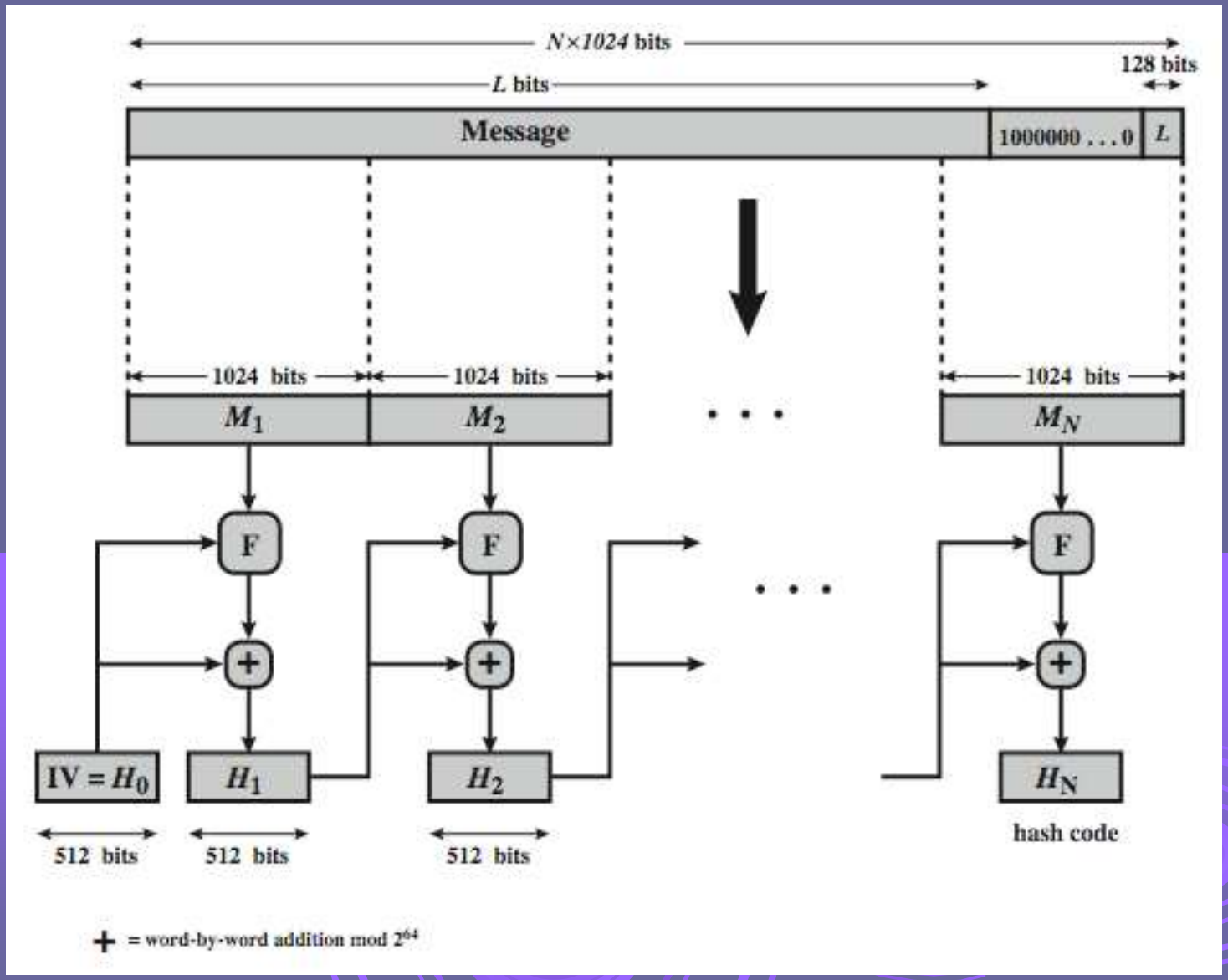
Revised Secure Hash Standard

- 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

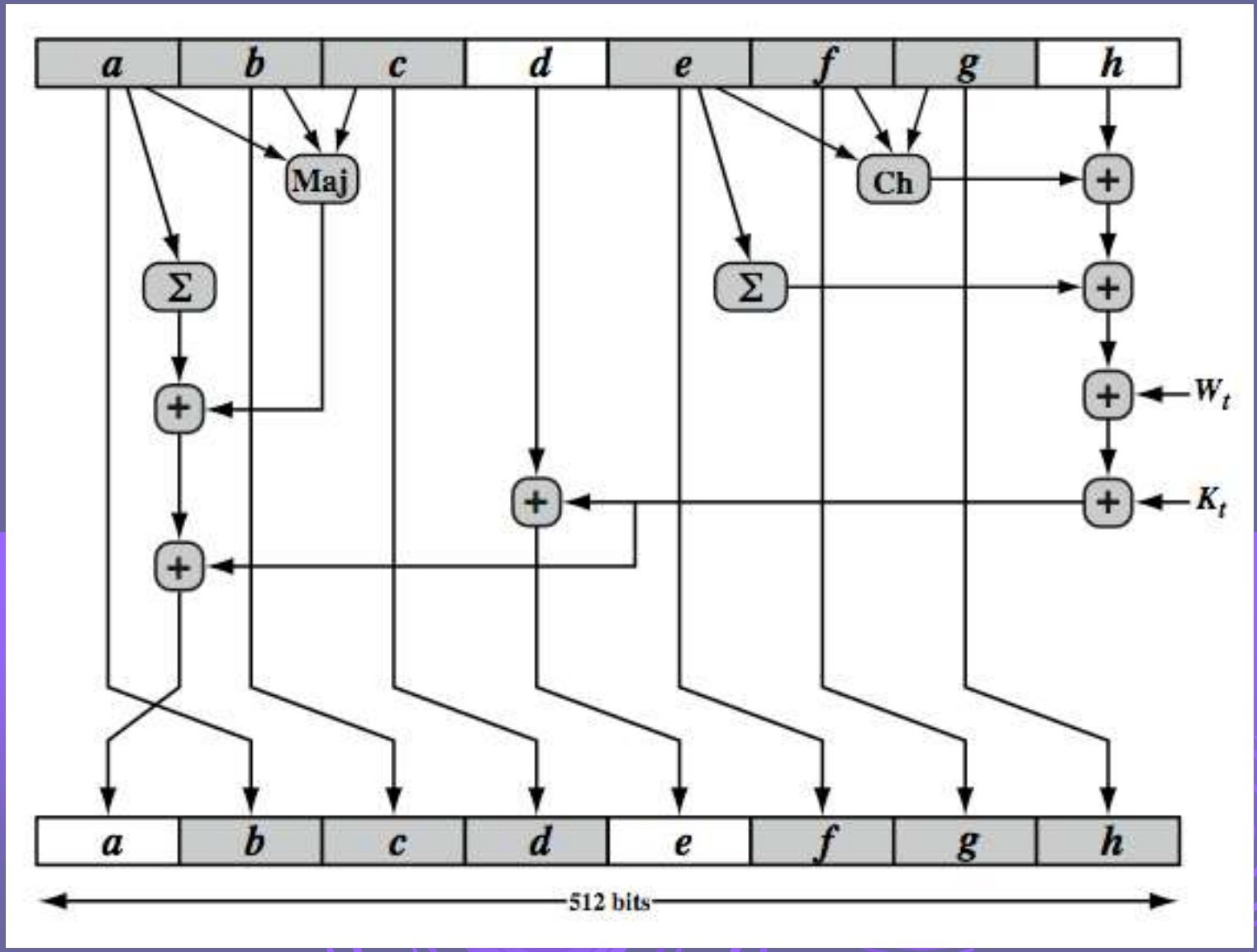
SHA-512 Overview



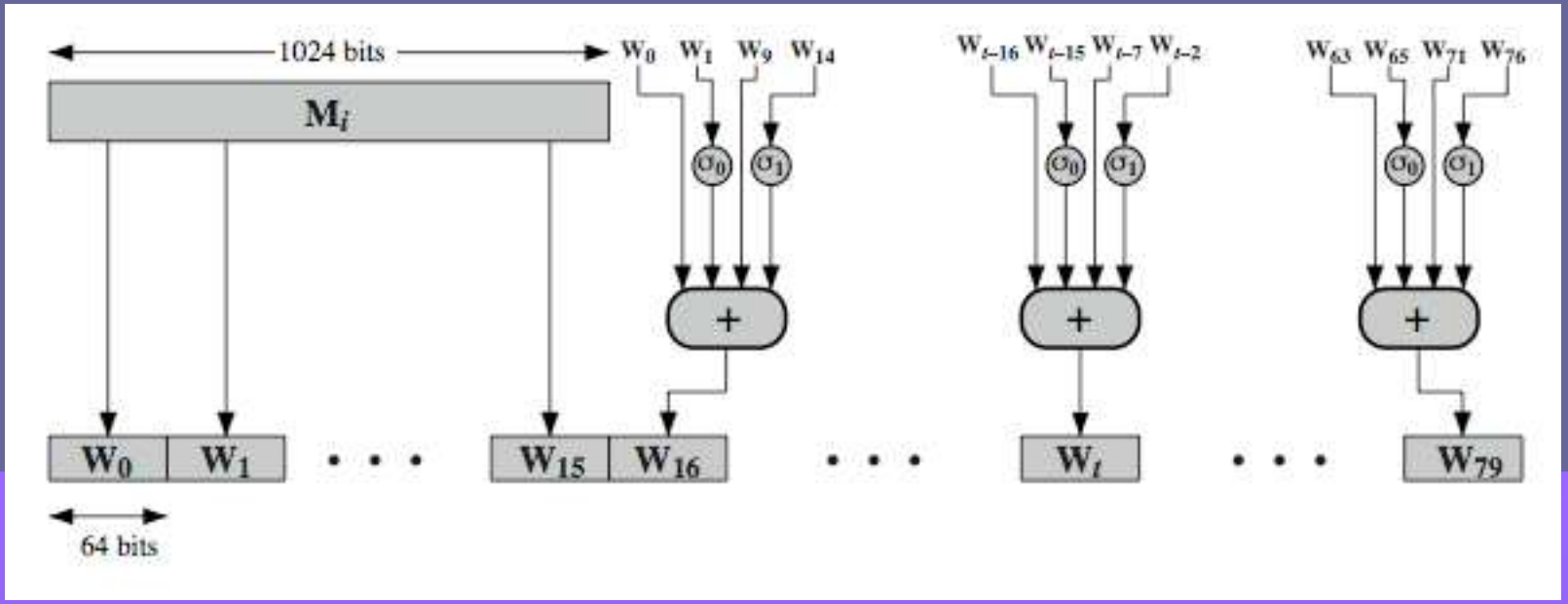
SHA-512 Compression Function

- 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

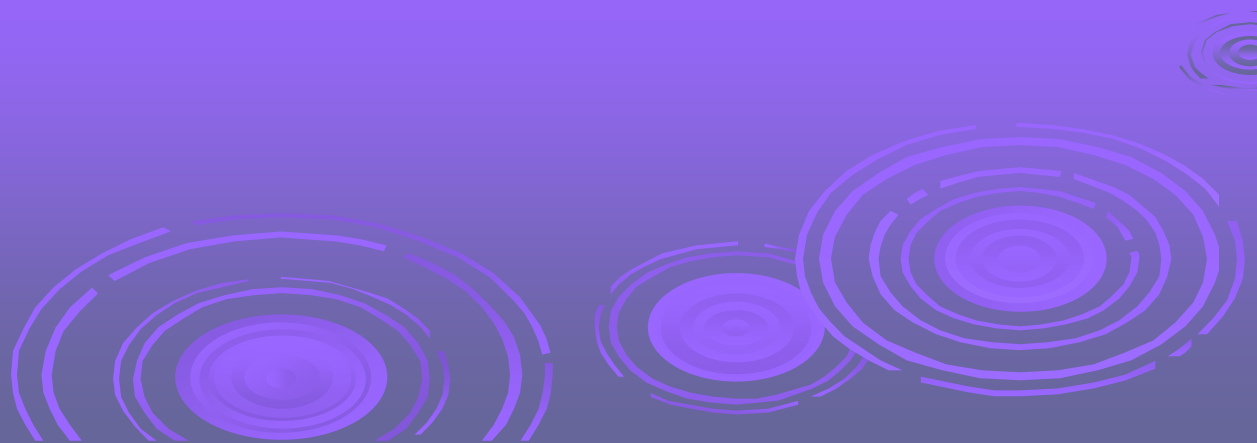
- SHA-1 not yet "broken"
  - but similar to broken MD5 & SHA-0
  - so considered insecure
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors so have concern
- NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function
- Keccak winner Oct 2012 – std in Q2,2014

SHA-3 Requirements

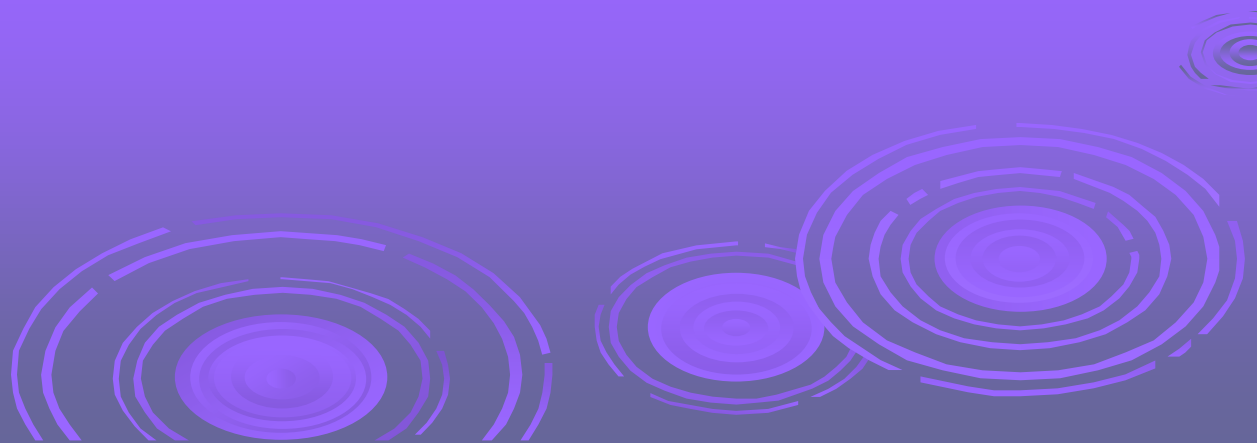
- replace SHA-2 with SHA-3 in any use
  - so use same hash sizes
- preserve the online nature of SHA-2
  - so must process small blocks (512 / 1024 bits)
- evaluation criteria
  - security close to theoretical max for hash sizes
  - cost in time & memory
  - characteristics: such as flexibility & simplicity

Summary

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



Cryptography and Network Security Chapter 13



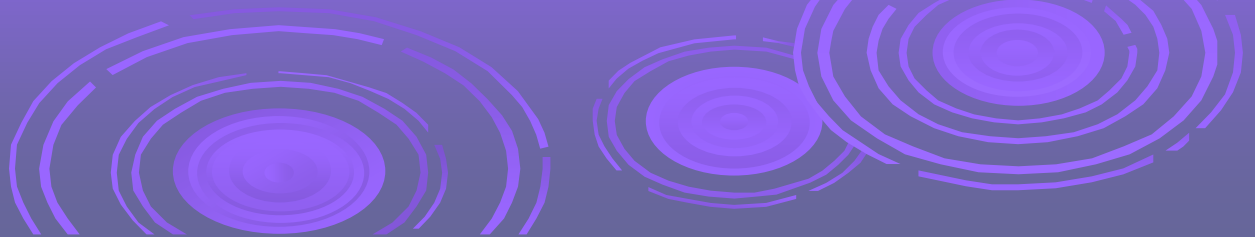
Chapter 13 – Digital Signatures

To guard against the baneful influence exerted by strangers is therefore an elementary dictate of savage prudence. Hence before strangers are allowed to enter a district, or at least before they are permitted to mingle freely with the inhabitants, certain ceremonies are often performed by the natives of the country for the purpose of disarming the strangers of their magical powers, or of disinfecting, so to speak, the tainted atmosphere by which they are supposed to be surrounded.

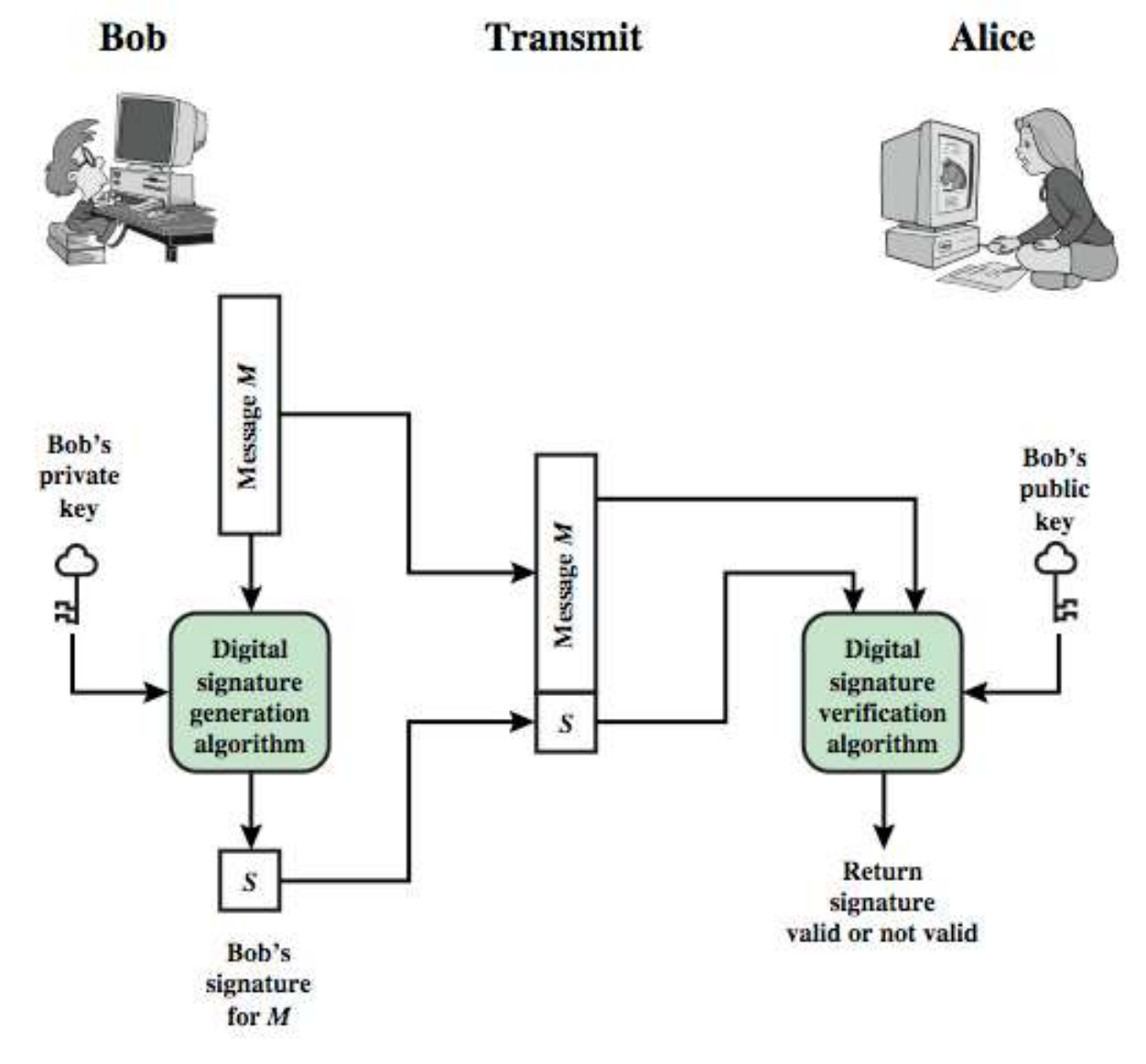
—*The Golden Bough*, Sir James George Frazer

Digital Signatures

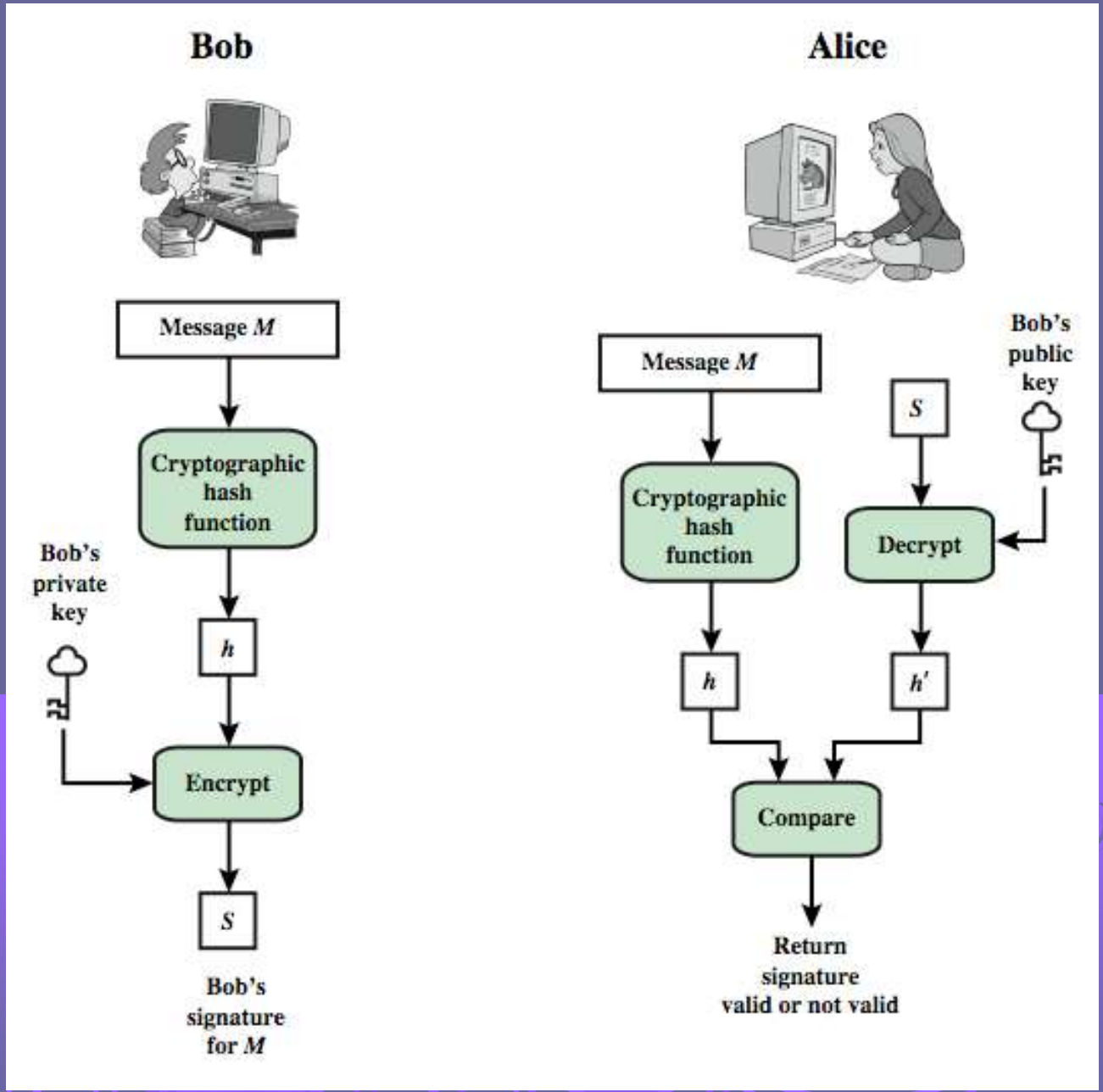
- have looked at message authentication
  - but does not address issues of lack of trust
- digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- hence include authentication function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



Digital Signature Model



Digital Signature Model



Attacks and Forgeries

- attacks
  - key-only attack
  - known message attack
  - generic chosen message attack
  - directed chosen message attack
  - adaptive chosen message attack
- break success levels
  - total break
  - selective forgery
  - existential forgery

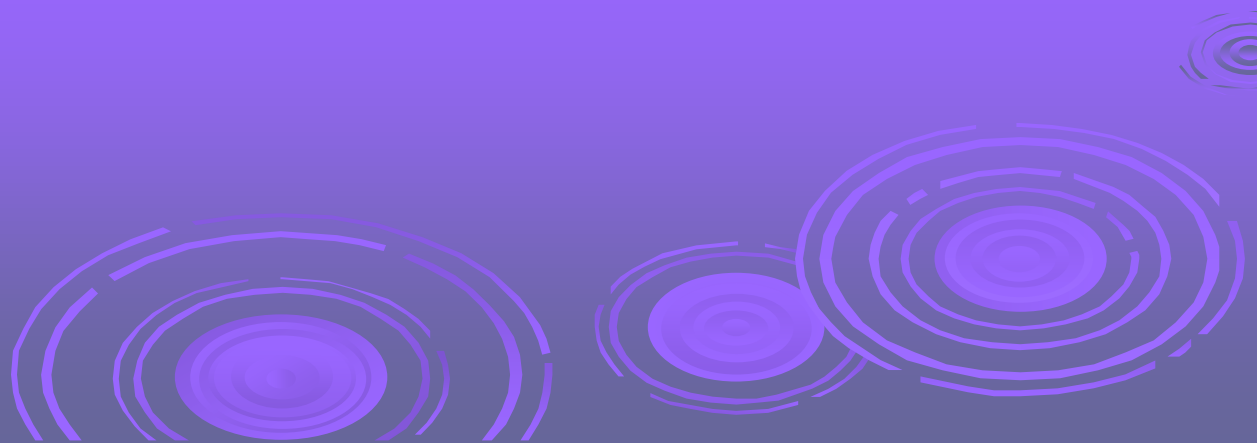
Key-only attack: C only knows A's public key.

• **Known message attack:** C is given access to a set of messages and their signatures.

• **Generic chosen message attack:** C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages.The attack is generic, because it does not depend on A's public key; the same attack is used against everyone.

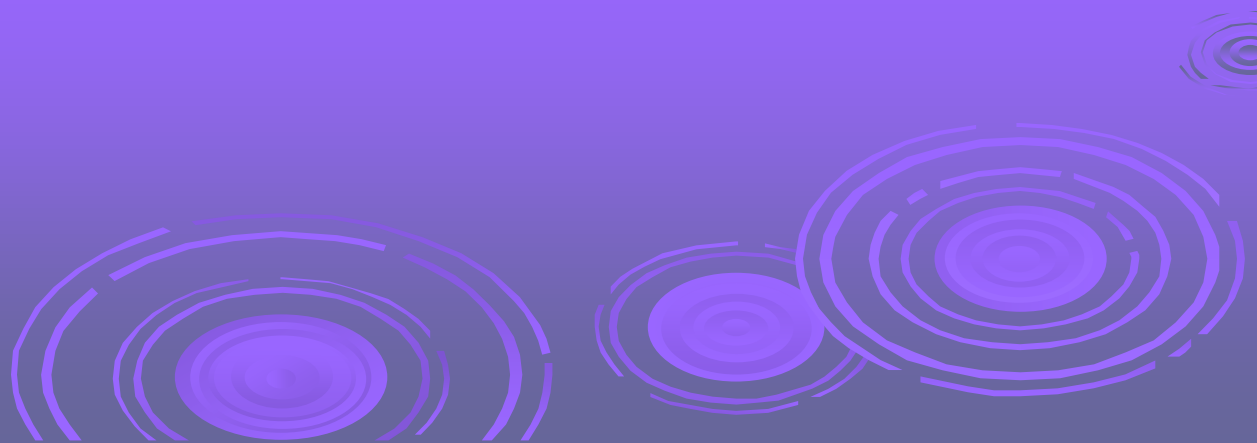
• **Directed chosen message attack:** Similar to the generic attack, except that the list of messages to be signed is chosen after C knows A's public key but before any signatures are seen.

• **Adaptive chosen message attack:** C is allowed to use A as an "oracle." This means the A may request signatures of messages that depend on previously obtained message–signature pairs.



Total break: C determines A's private key.

- **Universal forgery:** C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.
- **Selective forgery:** C forges a signature for a particular message chosen by C.
- **Existential forgery:** C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.



Digital Signature Requirements

- 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

ElGamal Digital Signatures

- use private key for encryption (signing)
- uses public key for decryption (verification)
- each user (e.g., A) generates their key
  - chooses a secret key (number): $1 < x_A < q-1$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$

ElGamal Digital Signature

- Alice signs a message M to Bob by computing
  - the hash $m = H(M)$, $0 \leq m \leq (q-1)$
  - chose random integer K with $1 \leq K \leq (q-1)$ and $\gcd(K, q-1) = 1$
  - compute temporary key: $S_1 = a^k \bmod q$
  - compute K^{-1} the inverse of $K \bmod (q-1)$
  - compute the value: $S_2 = K^{-1}(m - x_A S_1) \bmod (q-1)$
  - signature is: (S_1, S_2)
- any user B can verify the signature by computing
  - $V_1 = a^m \bmod q$
  - $V_2 = Y_A^{S_1} S_1^{S_2} \bmod q$
  - signature is valid if $V_1 = V_2$

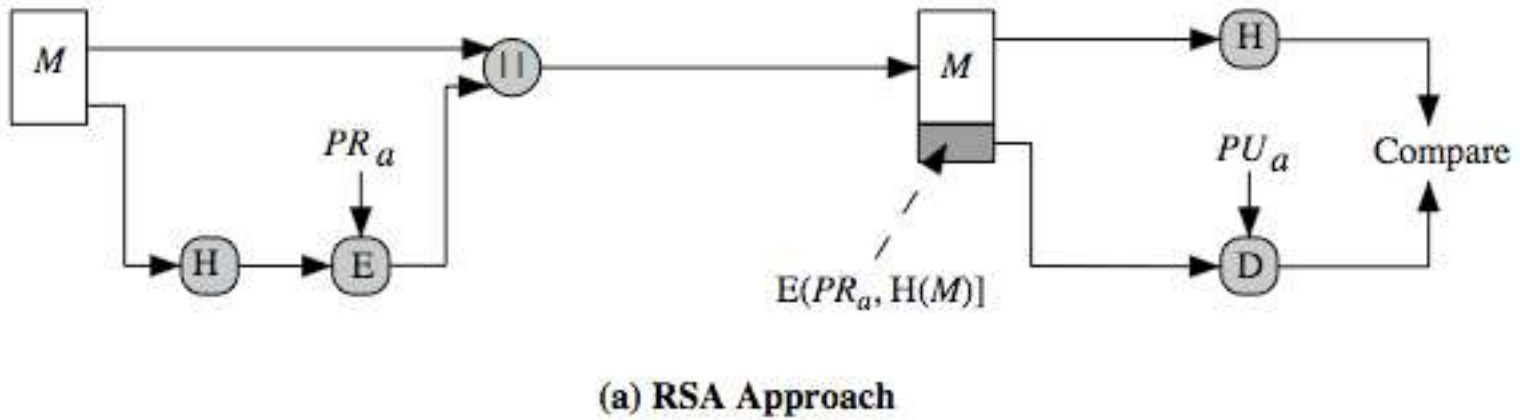
ElGamal Signature Example

- $q=19$ and $a=10$
- Alice computes her key:
  - A chooses $x_A=16$ & computes $y_A=10^{16} \bmod 19 = 4$
- Alice signs message with hash $m=14$ as $(3,4)$:
  - choosing random $K=5$ which has $\gcd(18,5)=1$
  - computing $S_1 = 10^5 \bmod 19 = 3$
  - finding $K^{-1} \bmod (q-1) = 5^{-1} \bmod 18 = 11$
  - computing $S_2 = 11(14-16.3) \bmod 18 = 4$
- any user B can verify the signature by computing
  - $V_1 = 10^{14} \bmod 19 = 16$
  - $V_2 = 4^3.3^4 = 5184 = 16 \bmod 19$
  - since $16 = 16$ signature is valid

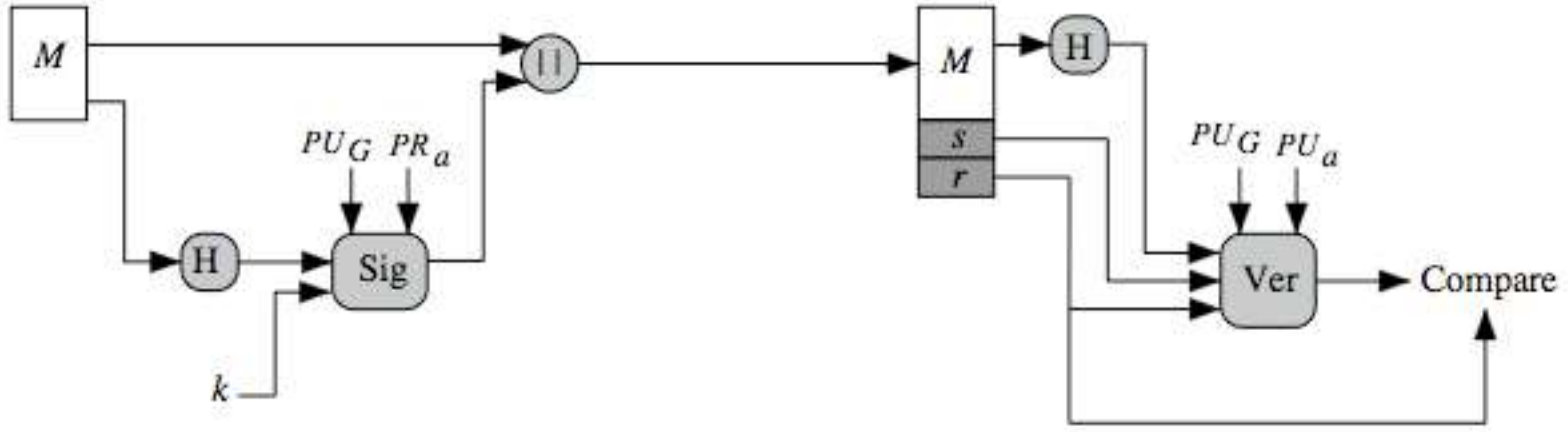
Digital Signature Standard (DSS)

- 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
- DSA is digital signature only unlike RSA
- is a public-key technique

DSS vs RSA Signatures



(a) RSA Approach



(b) DSS Approach

Digital Signature Algorithm (DSA)

- creates a 320 bit signature
- with 512-1024 bit security
- smaller and faster than RSA
- a digital signature scheme only
- security depends on difficulty of computing discrete logarithms
- variant of ElGamal & Schnorr schemes

DSA Key Generation

- have shared global public key values (p,q,g):
  - choose 160-bit prime number q
  - choose a large prime p with $2^{L-1} < p < 2^{L}$
    - where L= 512 to 1024 bits and is a multiple of 64
    - such that q is a 160 bit prime divisor of $(p-1)$
  - choose $g = h^{(p-1)/q}$
    - where $1<h<p-1$ and $h^{(p-1)/q} \bmod p > 1$
- users choose private & compute public key:
  - choose random private key: $x<q$
  - compute public key: $y = g^{x} \bmod p$

DSA Signature Creation

- to **sign** a message **M** the sender:
  - generates a random signature key **k, k<q**
  - nb. **k** must be random, be destroyed after use, and never be reused
- then computes signature pair:

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

$$s = [k^{-1}(H(M) + xr)] \bmod q$$

- sends signature **(r,s)** with message **M**

DSA Signature Verification

- having received M & signature **(r,s)**
- to **verify** a signature, recipient computes:

$$w = s^{-1} \bmod q$$

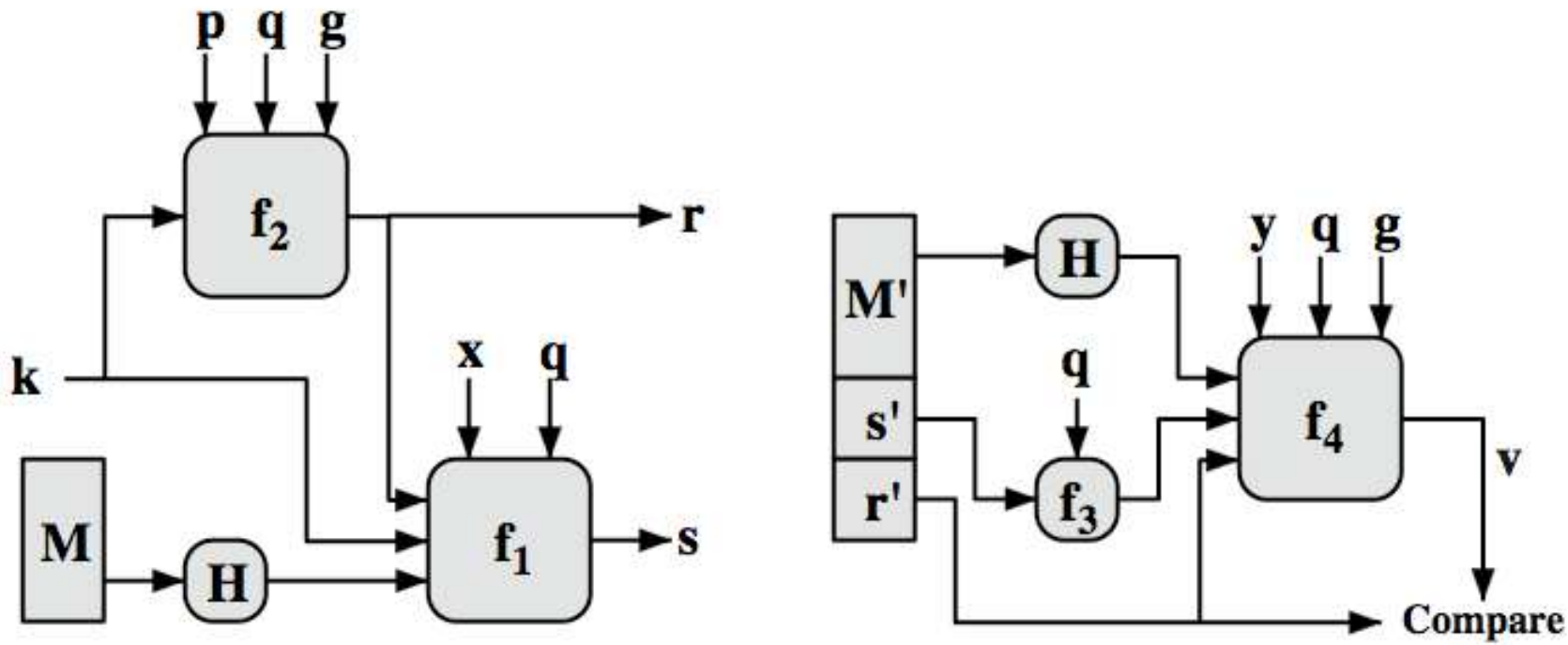
$$u1 = [H(M)w] \bmod q$$

$$u2 = (rw) \bmod q$$

$$v = [(g^{u1}\ y^{u2}) \bmod p] \bmod q$$

- if **v=r** then signature is verified
- see Appendix A for details of proof why

DSS Overview



$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

(a) Signing

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

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

(b) Verifying

Summary

- have discussed:
  - digital signatures
  - ElGamal & Schnorr signature schemes
  - digital signature algorithm and standard