CS 6530 Applied Cryptography

July-Nov 2025

Introduction to Cryptography and Data Security

18th August 2025 – S1, S2, 19th August 2025

Source: Cryptography and Network Security – William Stallings

Dr. Manikantan Srinivasan

Cryptographic Hash Functions

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

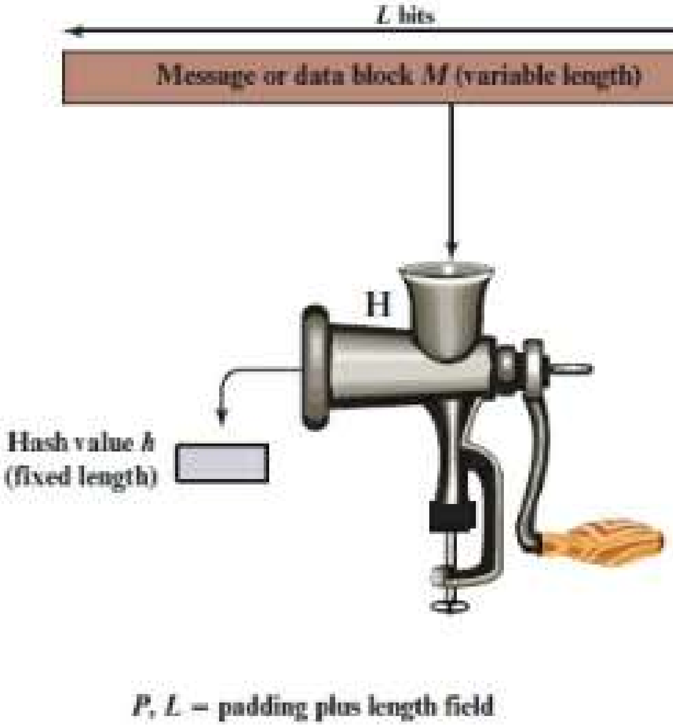
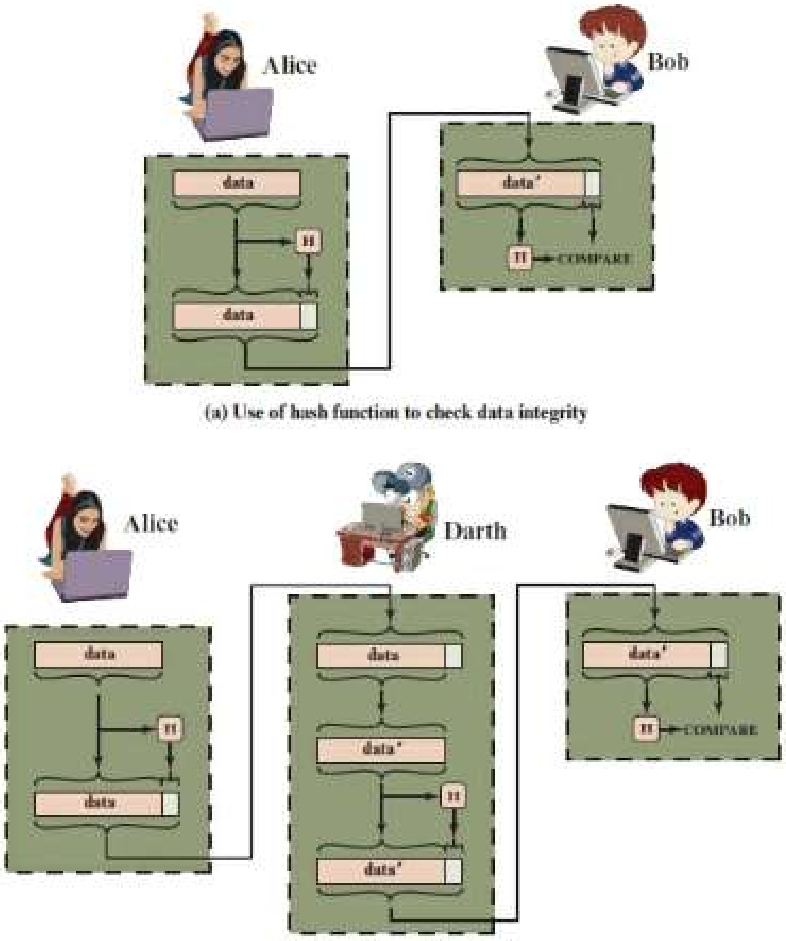

L bits
Message or data block M (variable length)
P, L
H
Hash value h
(fixed length)
P, L = padding plus length field

Figure 11.1 Cryptographic Hash Function; h = H(M)

Cryptographic Hash Function

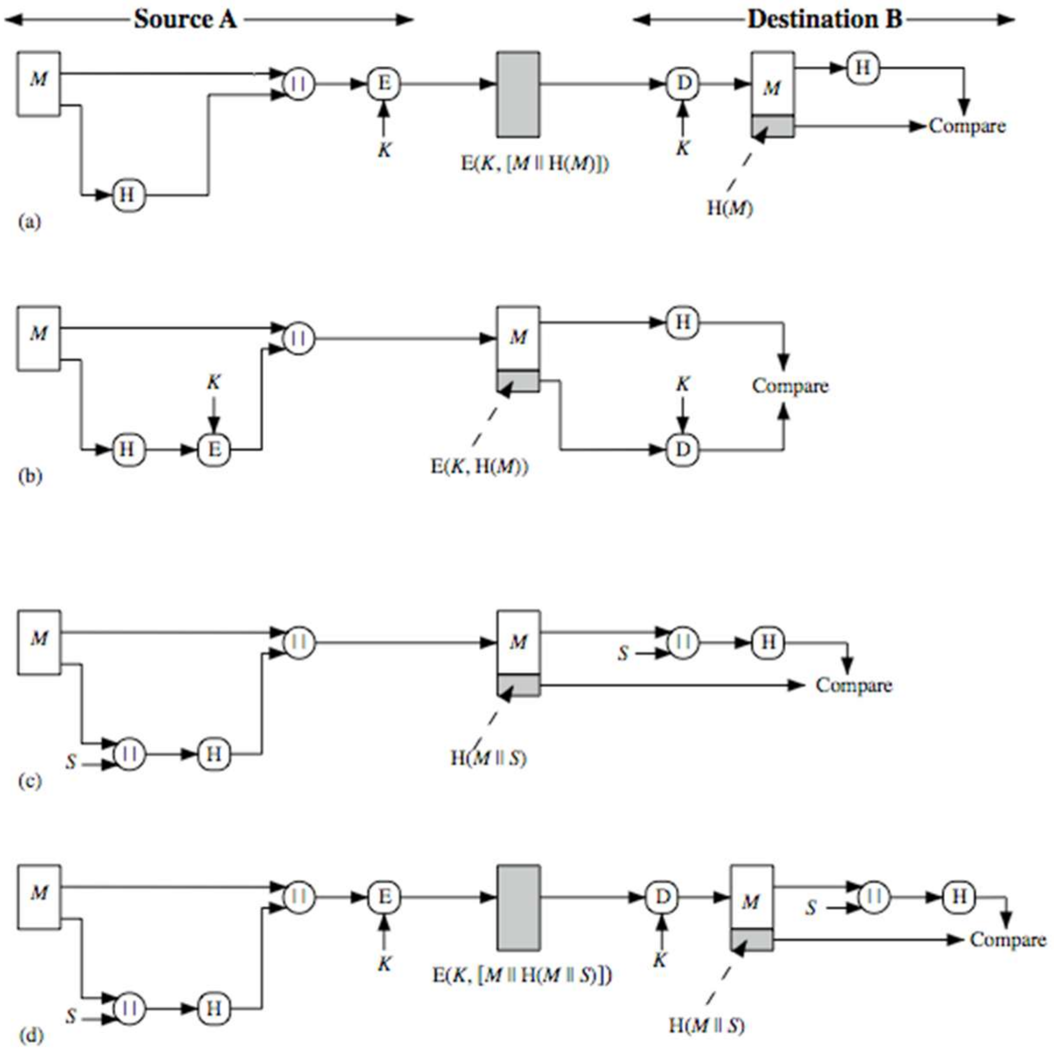


(a) Use of hash function to check data integrity

(b) Man-in-the-middle attack

The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver.

Hash Functions & Message Authent-ication



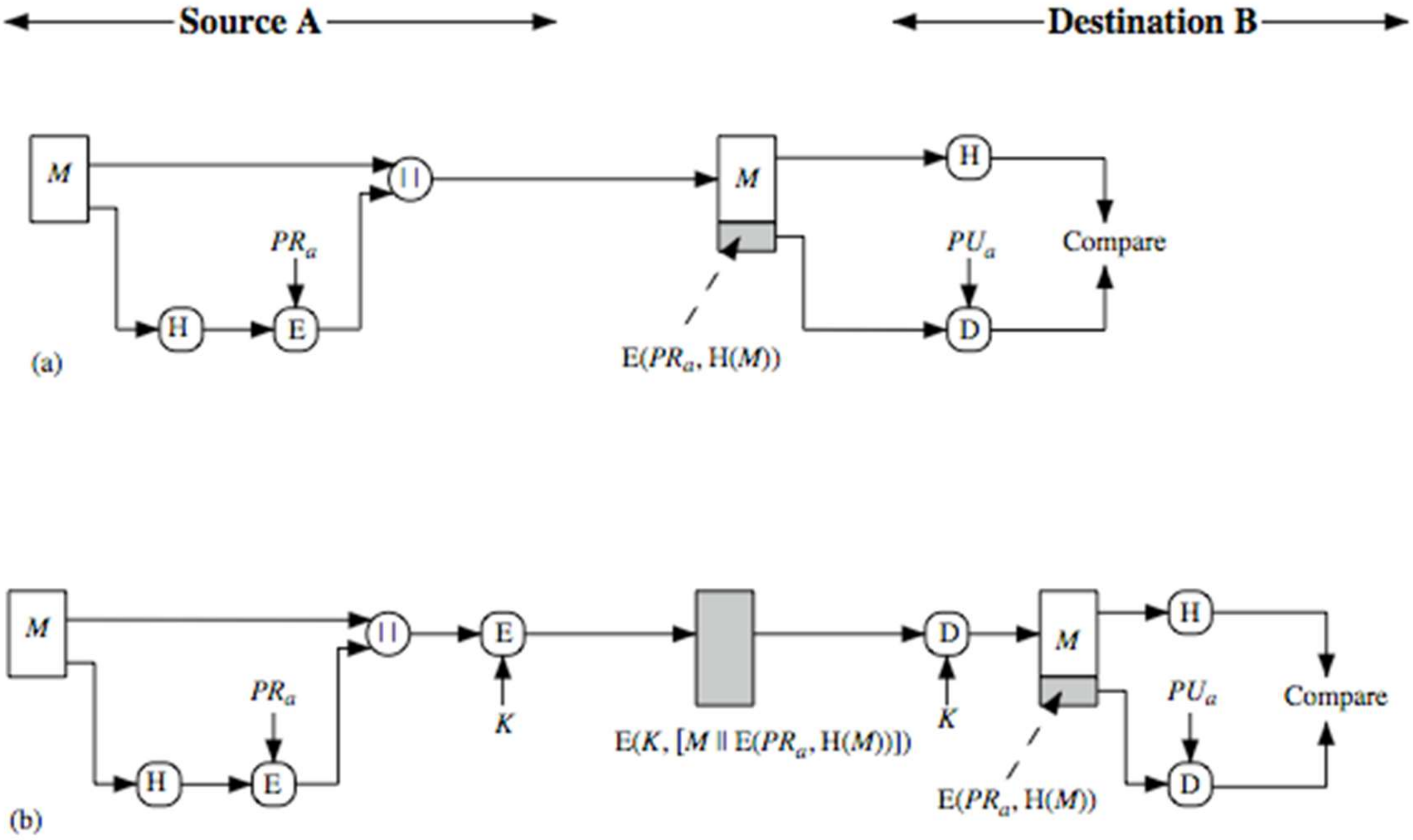
Message Authentication Code (MAC)

- Message authentication is achieved using a message authentication code (MAC), also known as a keyed hash function
- Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.
- A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.
- E(K, H(M)) is a function of a variable-length message M and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key.

Digital Signature

- The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user’s private key.
- Anyone who knows the user’s public key can verify the integrity of the message that is associated with the digital signature. In this case, an attacker who wishes to alter the message would need to know the user’s private key.
- The implications of digital signatures go beyond just message authentication.

Hash Functions & Digital Signatures



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)

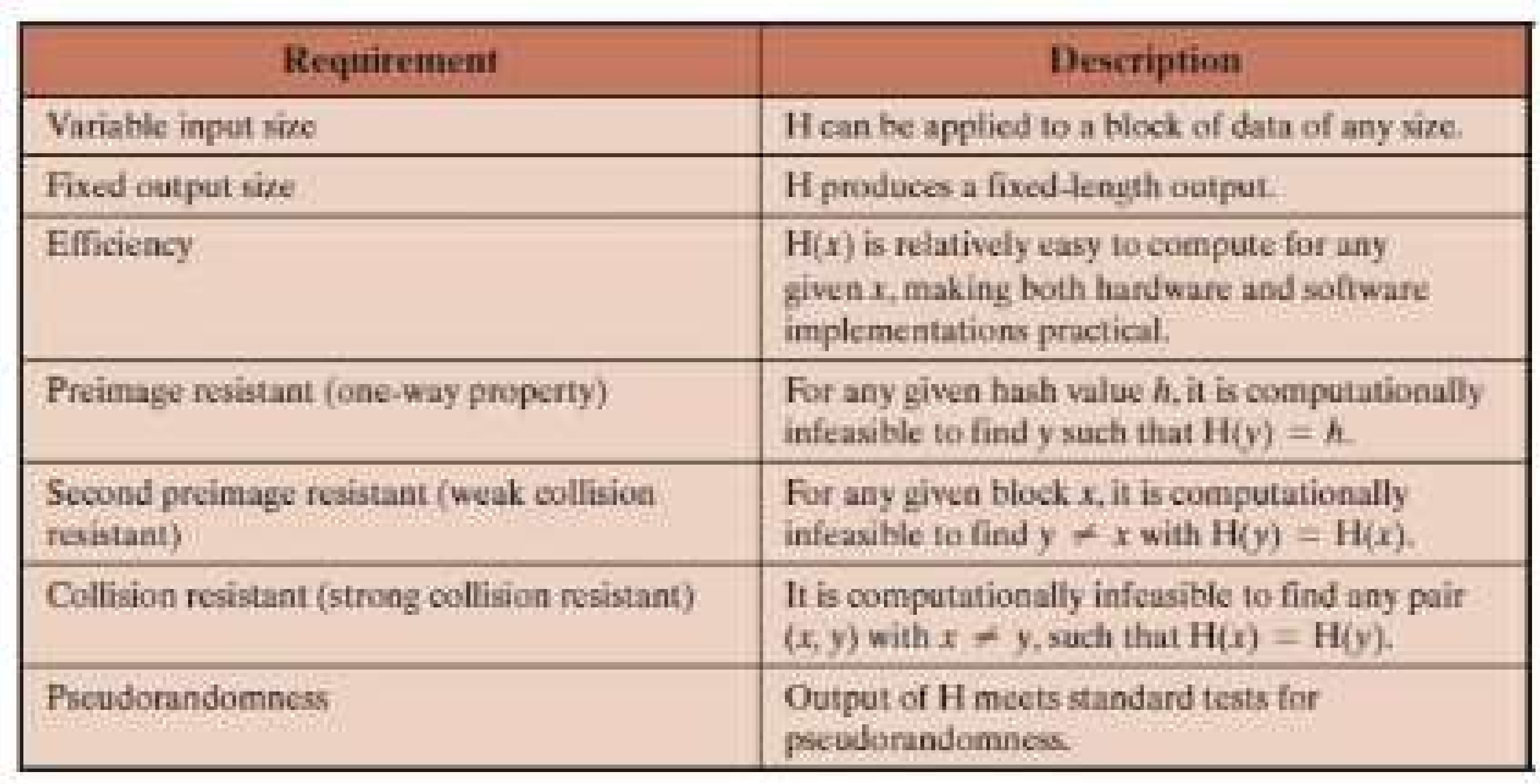
Two Simple Insecure Hash Functions

- consider two simple insecure hash functions
- bit-by-bit exclusive-OR (XOR) of every block
  - $C_i = b_{i1}\ xor\ b_{i2}\ xor \ldots xor\ b_{im}$
  - 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

A Definition

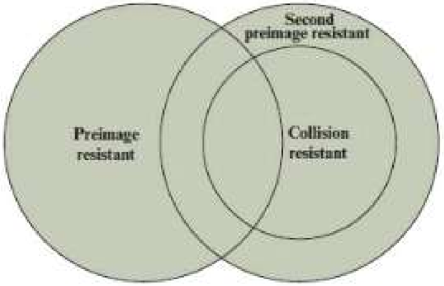
- For a hash value h = H(x), we say that x is the **preimage** of h. That is, x is a data block whose hash value, using the function H, is h.
- Because H is a many-to-one mapping, for any given hash value h, there will in general be multiple preimages.
- A **collision** occurs if we have x ≠ y and H(x) = H(y).
- Because we are using hash functions for data integrity, collisions are clearly undesirable.

Hash Function Requirements

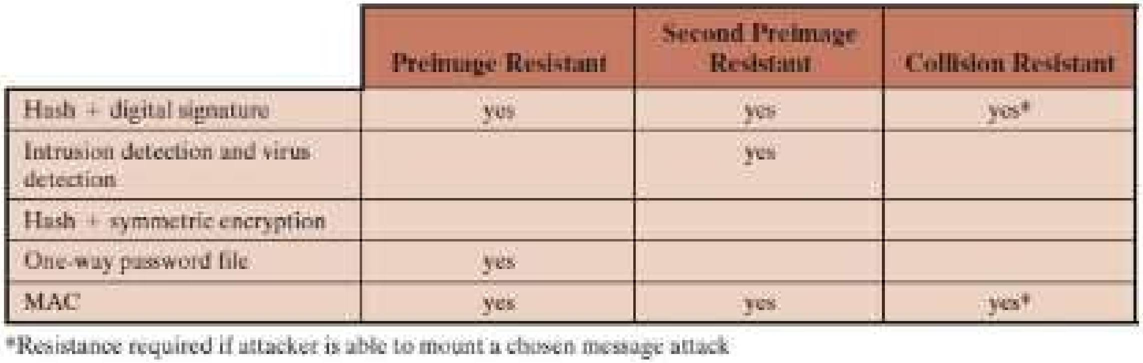


Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	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) with $x \neq y$, such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Hash Function Requirements



Relationship Among Hash Function Properties



	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	yes	yes	yes*
Intrusion detection and virus detection		yes	
Hash + symmetric encryption			
One-way password file	yes		
MAC	yes	yes	yes*

*Resistance required if attacker is able to mount a chosen message attack

Hash Function Resistance Properties Required for Various Data Integrity Applications

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

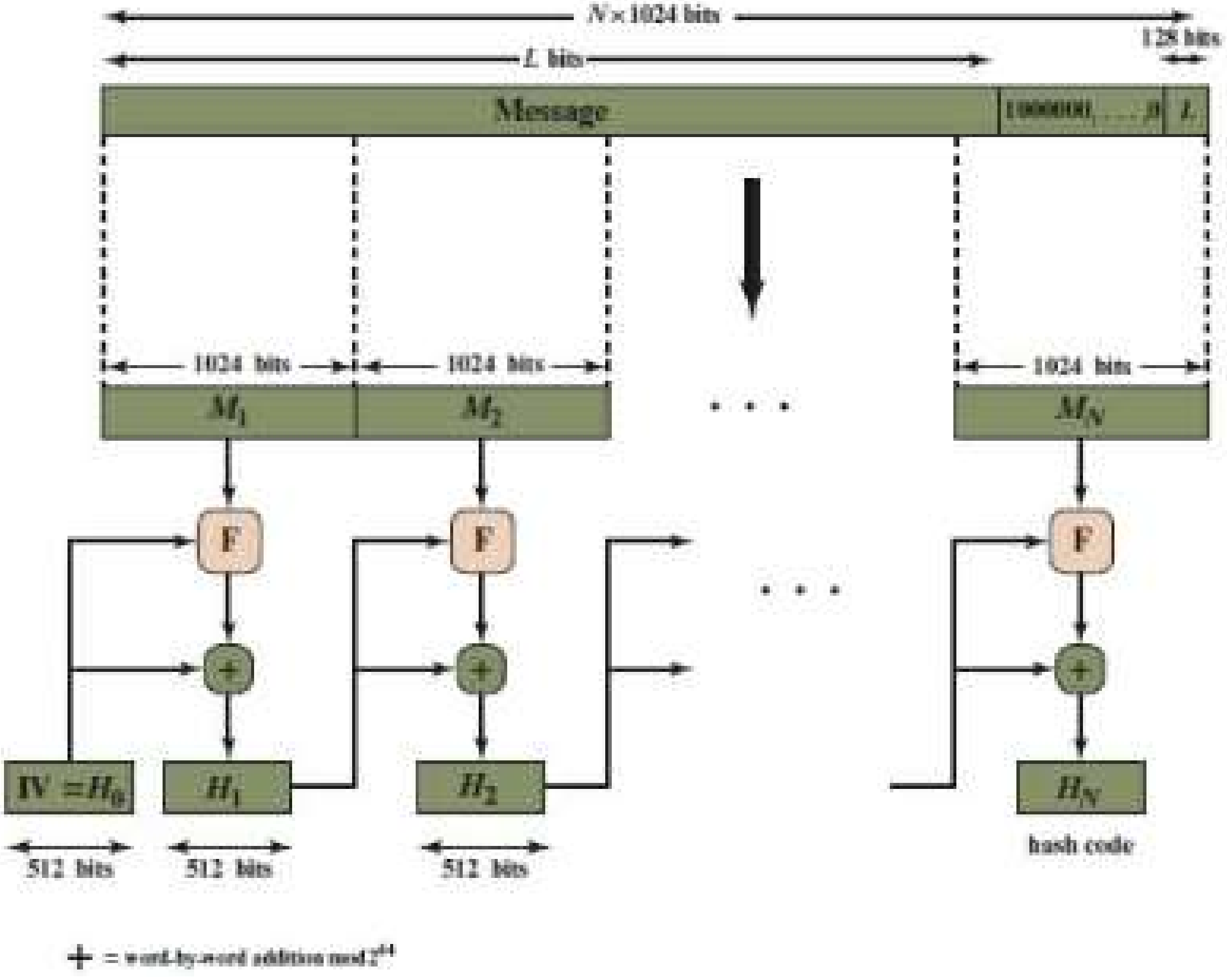
Revised Secure Hash Standard

- 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



The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest.

The input is processed in 1024-bit blocks.

Step 1 Append padding bits.

Step 2 Append length.

Step 3 Initialize hash buffer.

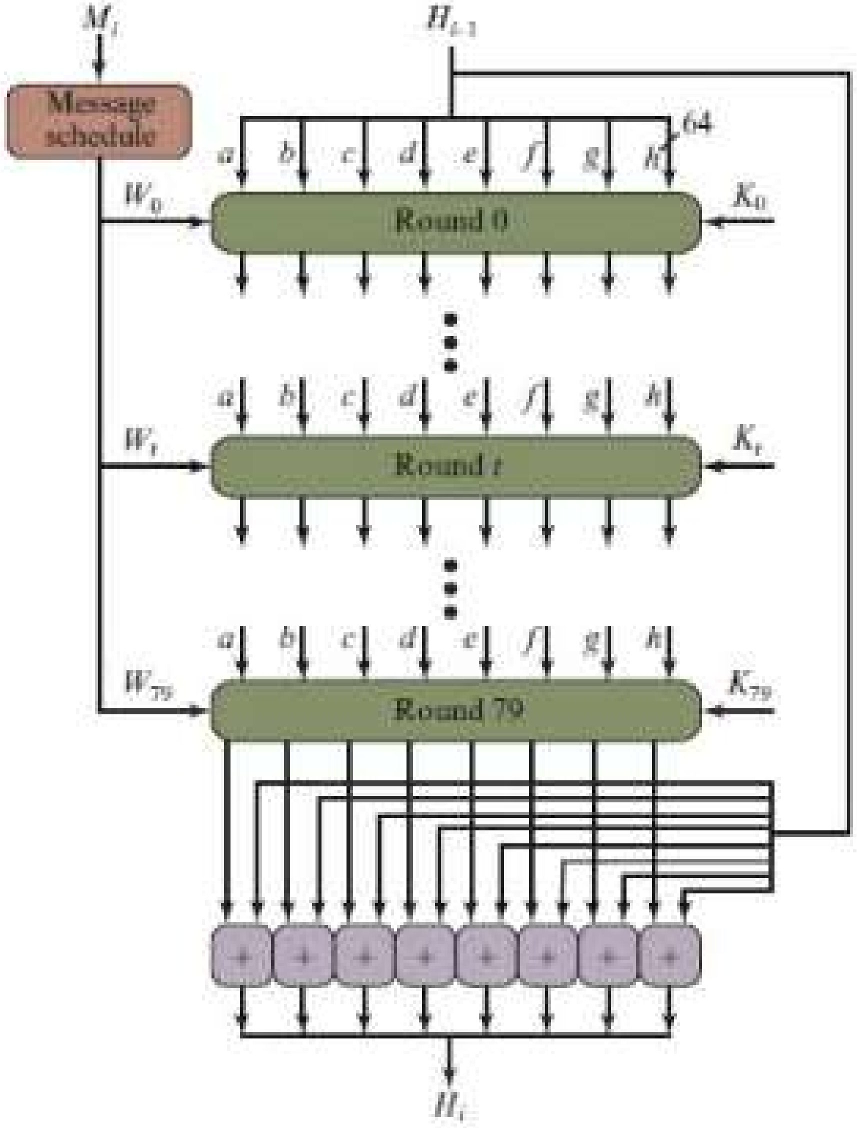
Step 4 Process message in 1024-bit (128-byte) blocks.

Message Digest Generation Using SHA-512

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 Overview

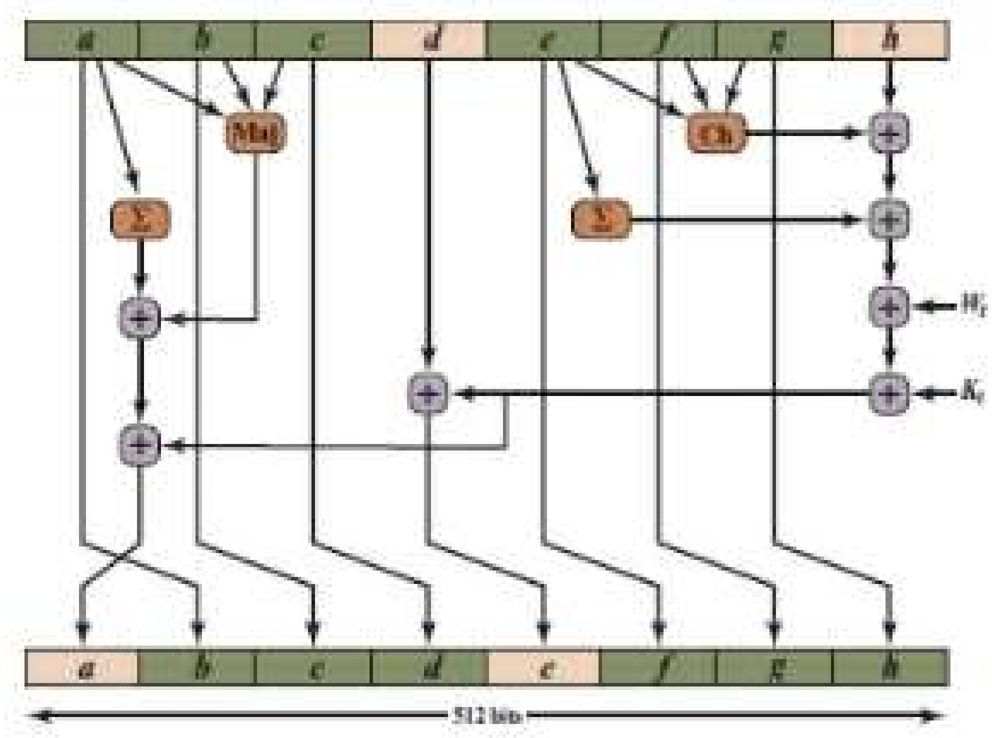


SHA-512 Processing of a Single 1024-Bit Block

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest.
The input is processed in 1024-bit blocks.

SHA-512 Round Function

Elementary SHA-512 Operation (single round)



Each round is defined by the following set of equations:

$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum\nolimits_1^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\sum\nolimits_0^{512} a\right) + \text{Maj}(a, b, c)$$

$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

Hash Buffer
Register a
Register b
Register c
Register d
Register e
Register f
Register g
Register h

Initialization Vector

a = 0x6A09E667F3BCC908	b = 0xBB67AE8584CAA73B
c = 0x3C6EF372FE94F82B	d = 0xA54FF53A5F1D36F1
e = 0x510E527FADE682D1	f = 0x9B05688C2B3E6C1F
g = 0x1F83D9ABFB41BD6B	h = 0x5BE0CD19137E2179

first 64 bits of the fractional parts of the square roots of the first 8 prime numbers (2,3,5,7,11,13,17,19)

t = step number; $0 \le t \le 79$

$\text{Ch}(e, f, g)$ = $(e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$
the conditional function: If e then f else g

$\text{Maj}(a, b, c)$ = $(a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
the function is true only of the majority (two or three) of the arguments are true

$(\Sigma_0^{512} a)$ = $\text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$

$(\Sigma_1^{512} e)$ = $\text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$

$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

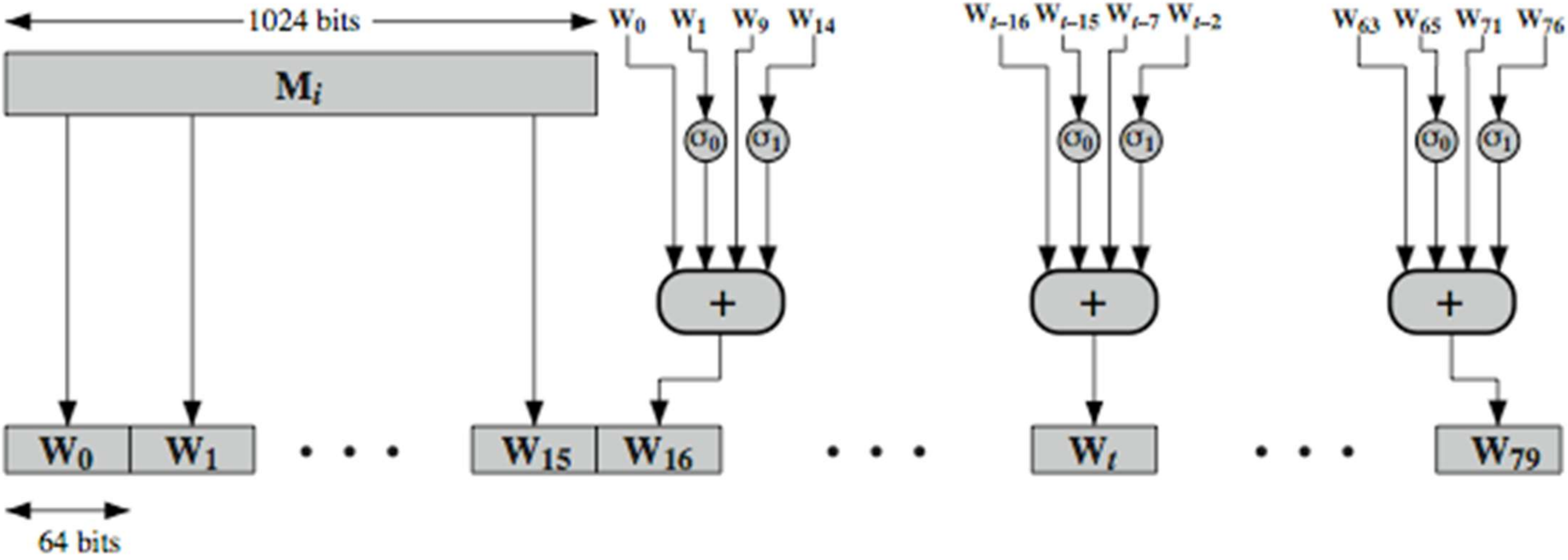
K_t = first 64 bits from the fractional part of the cube roots of the first 80 prime numbers

W_t = a 64-bit word derived from the current 1024-bit input block

K_t = a 64-bit additive constant

+ = addition modulo 2^{64}

SHA-512 Round Function


1024 bits
M_i
W_0 W_1 W_9 W_{14}
W_{t-16} W_{t-15} W_{t-7} W_{t-2}
W_{63} W_{65} W_{71} W_{76}
σ_0
σ_1
+
W_0
W_1
• • •
W_{15}
W_{16}
W_t
W_{79}
64 bits

Wt are derived from the 1024-bit message. The first 16 values of Wt are taken directly from the 16 words of the current block. The remaining values are defined as a function of the earlier values using ROTates, SHIFTs and XORs as shown. The function elements are:

∂0(x) = ROTR(x,1) XOR ROTR(x,8) XOR SHR(x,7)

∂1(x) = ROTR(x,19) XOR ROTR(x,61) XOR SHR(x,6)

SHA-3

- SHA-1 is broken
  - indicated in 2011.
- 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

Public Key Cryptography and RSA

Private-Key Cryptography

- traditional private/secret/single key cryptography uses one key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is symmetric, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

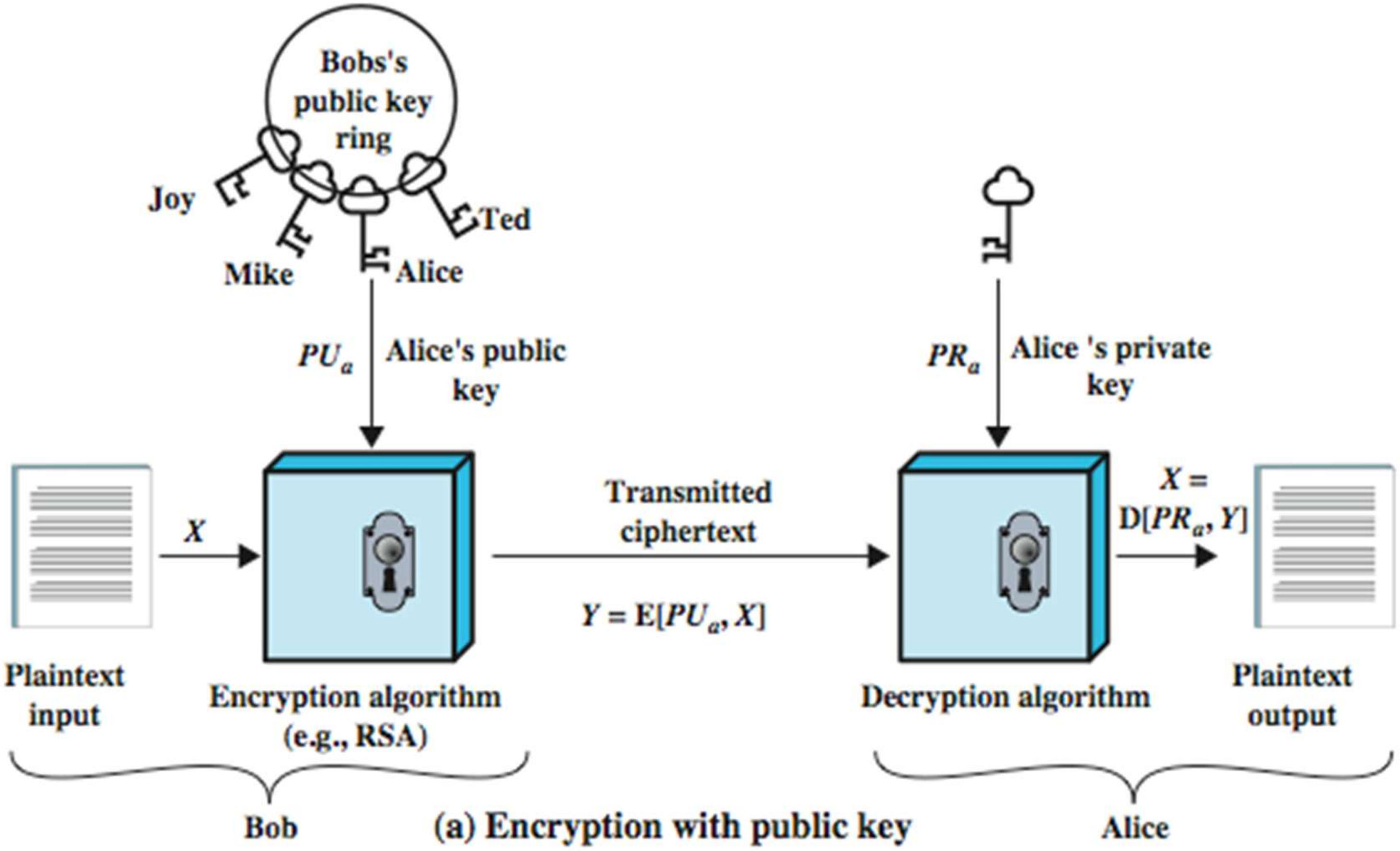
Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community

Public-Key Cryptography

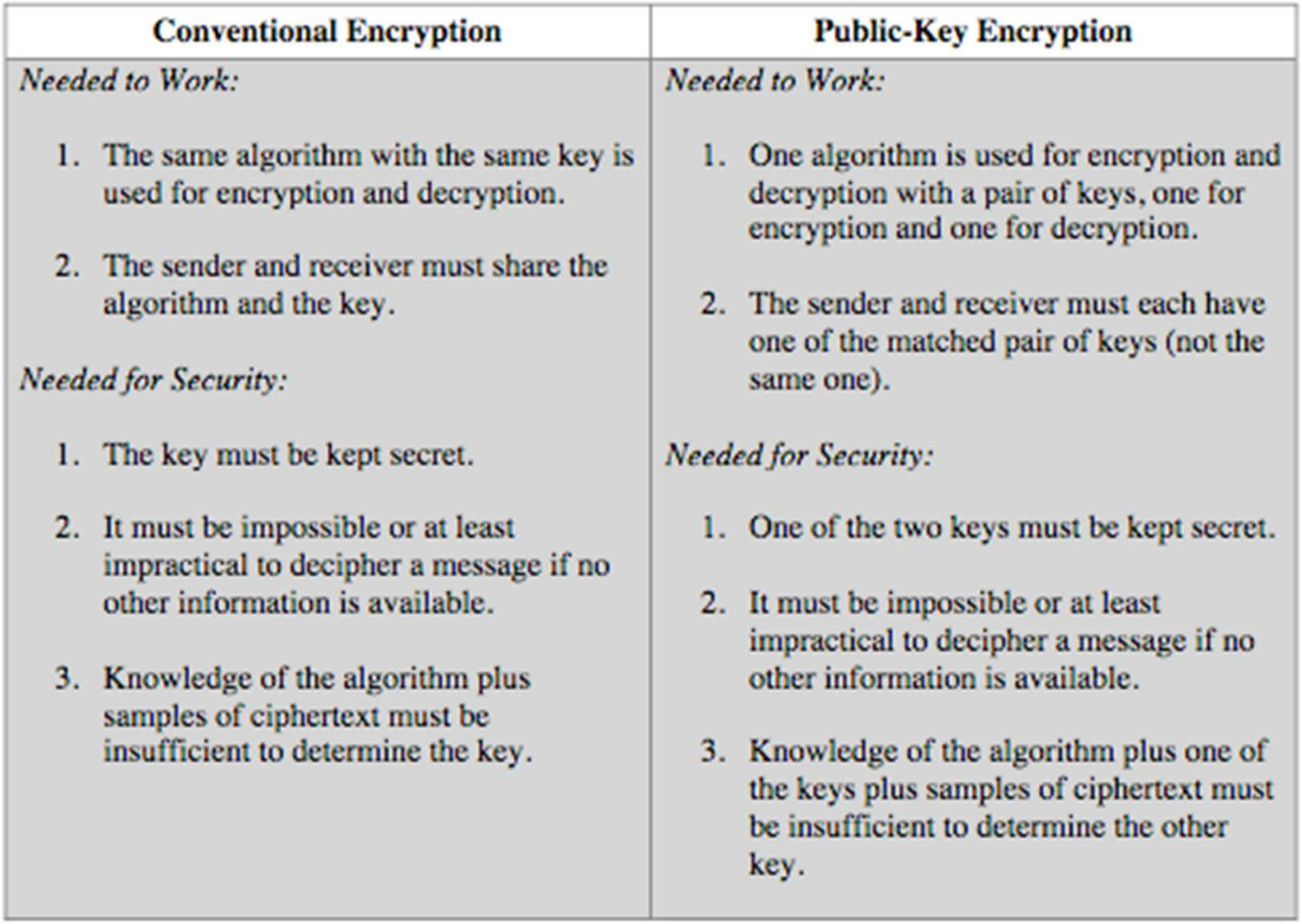
- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a related **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- **infeasible to determine private key from public**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

Public-Key Cryptography



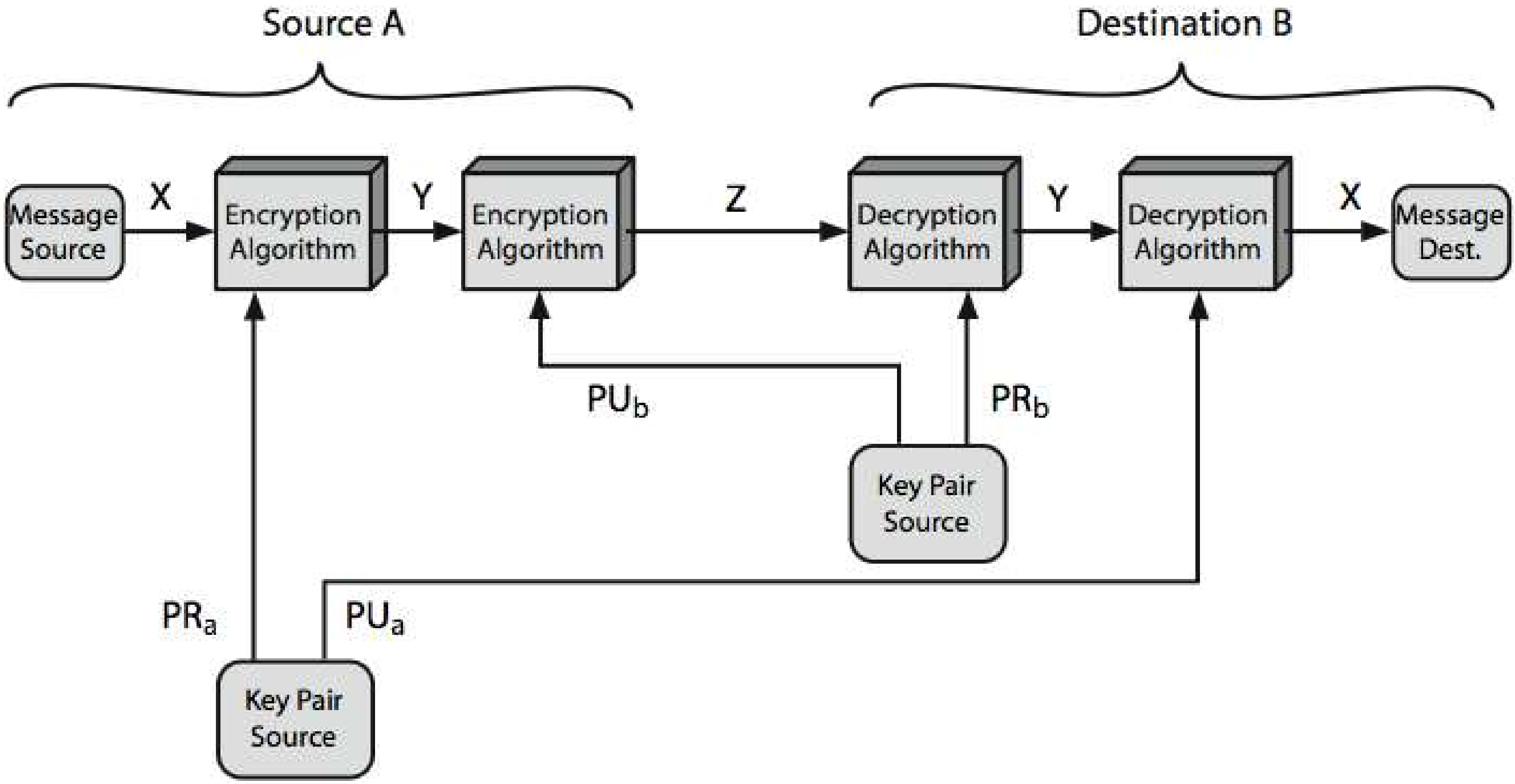
(a) Encryption with public key

Symmetric vs Public-Key



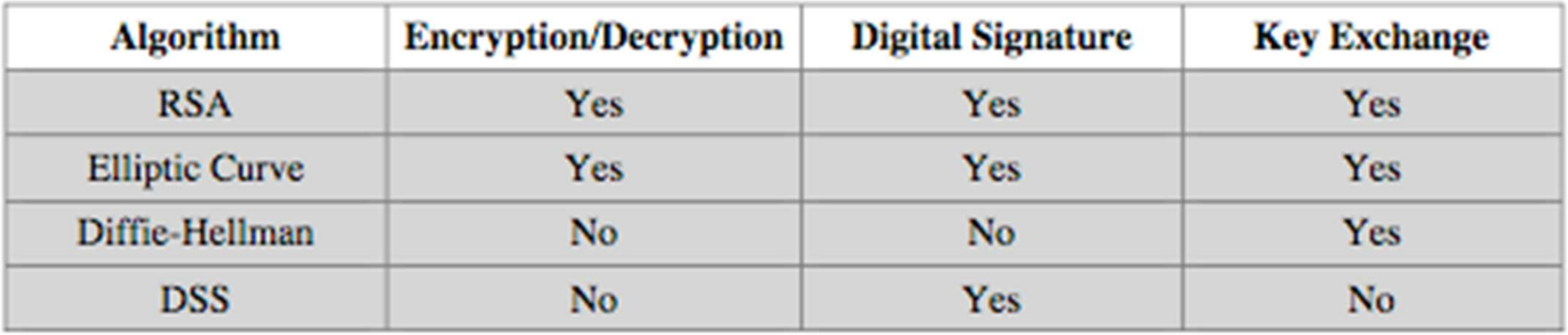
Conventional Encryption	**Public-Key Encryption**
Needed to Work:	*Needed to Work:*
1. The same algorithm with the same key is used for encryption and decryption.	1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.
2. The sender and receiver must share the algorithm and the key.	2. The sender and receiver must each have one of the matched pair of keys (not the same one).
Needed for Security:	*Needed for Security:*
1. The key must be kept secret.	1. One of the two keys must be kept secret.
2. It must be impossible or at least impractical to decipher a message if no other information is available.	2. It must be impossible or at least impractical to decipher a message if no other information is available.
3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Public-Key Cryptosystems



Public-Key Applications

- can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- some algorithms are suitable for all uses, others are specific to one



Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Public-Key Requirements

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)
- these are formidable requirements which only a few algorithms have satisfied

Public-Key Requirements

- need a trapdoor one-way function
- one-way function has
  - $Y = f(X)$ easy
  - $X = f^{-1}(Y)$ infeasible
- a trap-door one-way function has
  - $Y = f_k(X)$ easy, if k and X are known
  - $X = f_k^{-1}(Y)$ easy, if k and Y are known
  - $X = f_k^{-1}(Y)$ infeasible, if Y known but k not known
- a practical public-key scheme depends on a suitable trap-door one-way function

Security of Public Key Schemes

- like private key schemes brute force exhaustive search attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalyse) problems
- more generally the hard problem is known, but is made hard enough to be impractical to break
- requires the use of very large numbers
- hence is slow compared to private key schemes

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
  - nb. 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 En/decryption

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

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,n}

Why RSA Works

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

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

$$= M^1.(1)^k = M^1 = M \bmod n$$

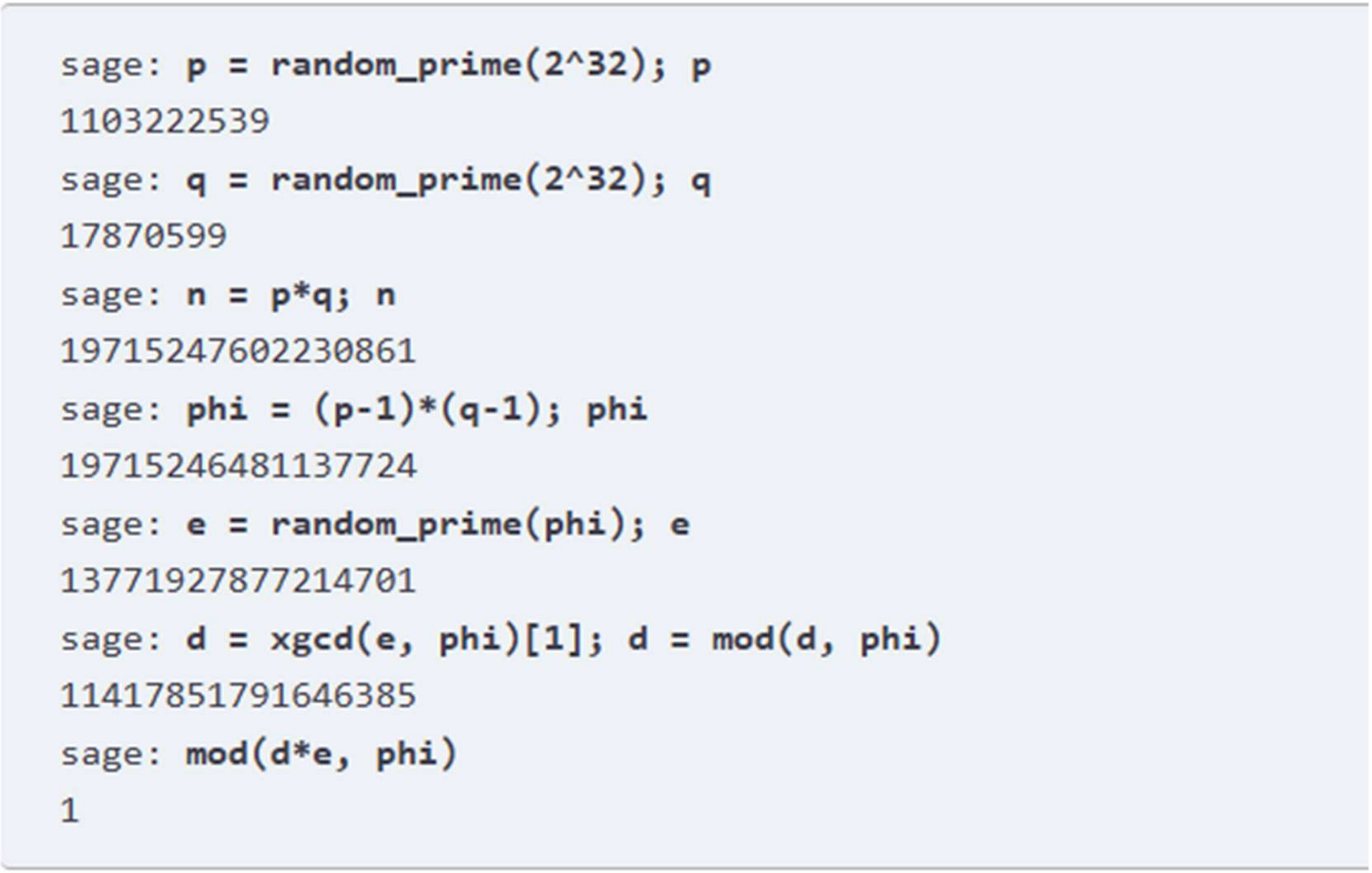
RSA Example - Key Setup

1. Select primes: $p=17$ & $q=11$
2. Calculate $n = pq = 17$ x $11=187$
3. Calculate $\varnothing(n)=(p-1)(q-1)=16x10=160$
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 $23x7=161= 1x160+1$
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:
  - $M = 11^{23} \bmod 187 = 88$

RSA Another Example



```
sage: p = random_prime(2^32); p
1103222539
sage: q = random_prime(2^32); q
17870599
sage: n = p*q; n
19715247602230861
sage: phi = (p-1)*(q-1); phi
19715246481137724
sage: e = random_prime(phi); e
13771927877214701
sage: d = xgcd(e, phi)[1]; d = mod(d, phi)
11417851791646385
sage: mod(d*e, phi)
1
```

Listing 10-1: Generating RSA parameters using SageMath

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. $7^5 = 7^4.7^1 = 3.7 = 10 \text{ mod } 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \text{ mod } 11$

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
```

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

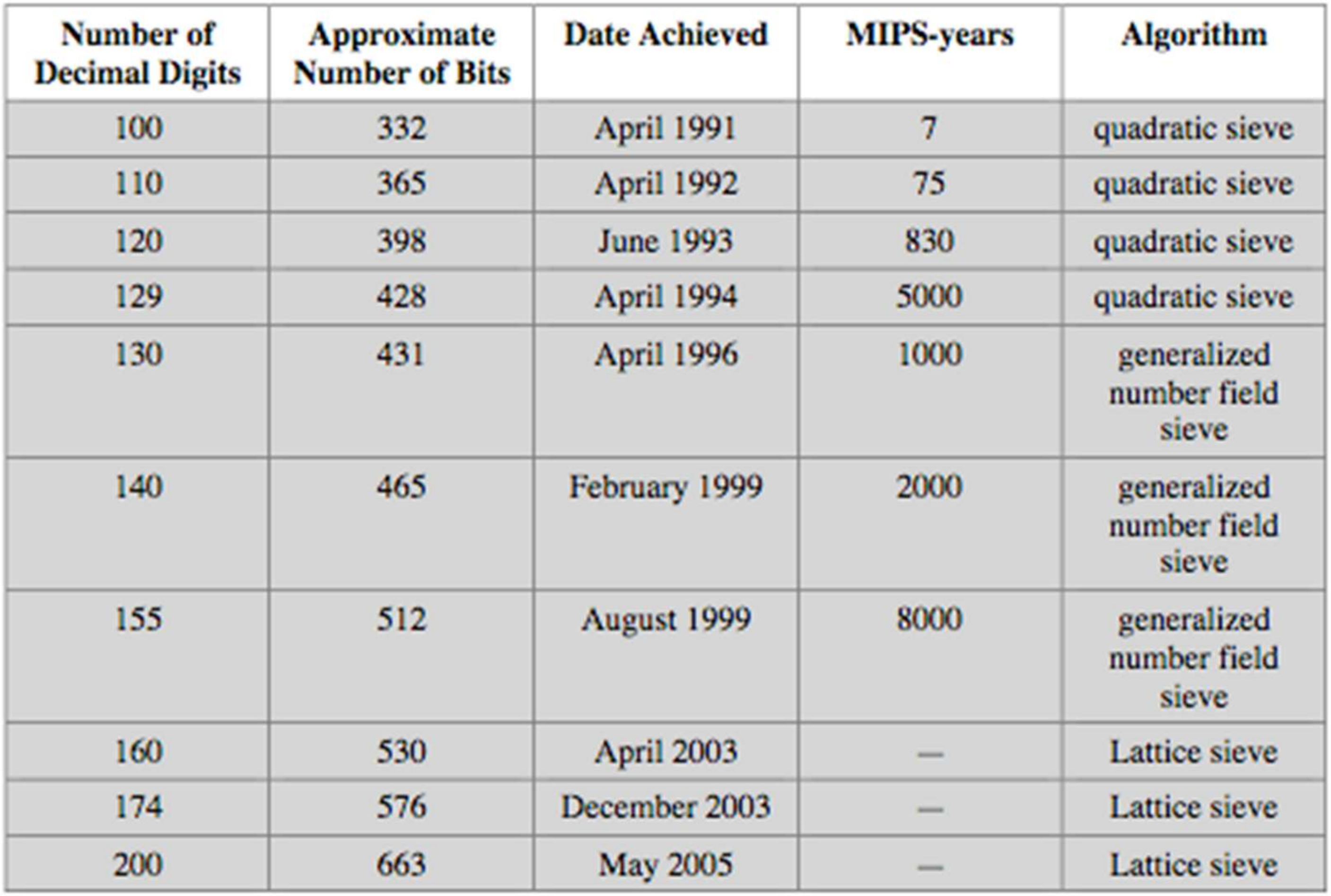
RSA Security

- 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

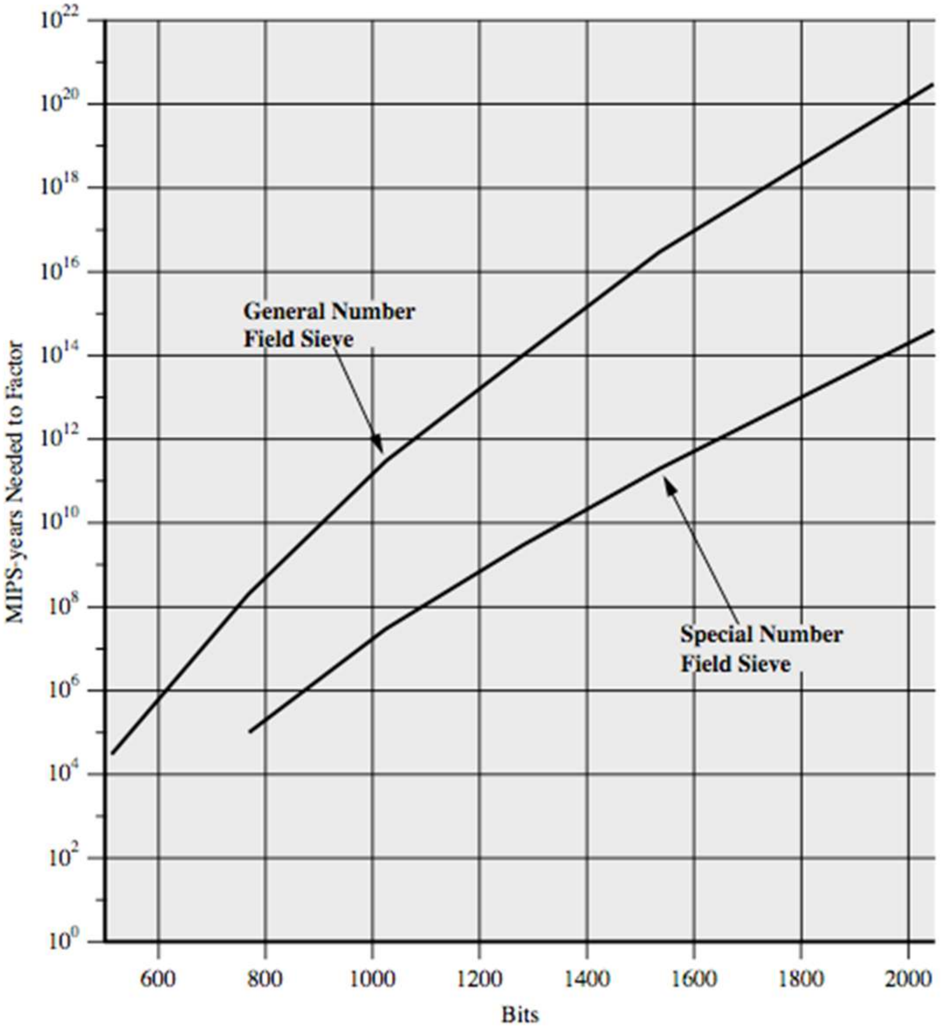
- mathematical approach takes 3 forms:
  - factor n=p.q, hence compute ø(n) and then d
  - determine ø(n) directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with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

Progress in Factoring



Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve

Progress in Factoring



Timing Attacks

- developed by Paul Kocher in mid-1990’s
- exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

Chosen Ciphertext Attacks

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- attackers chooses ciphertexts & gets decrypted plaintext back
- choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- can counter with random pad of plaintext
- or use Optimal Asymmetric Encryption Padding (OASP)

Summary

- have considered:
  - principles of public-key cryptography
  - RSA algorithm, implementation, security

Diffie Hellman Key Exchange

Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root mod q
- each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their public key: $Y_A = a^{x_A} \bmod q$
- each user makes public that key Y_A

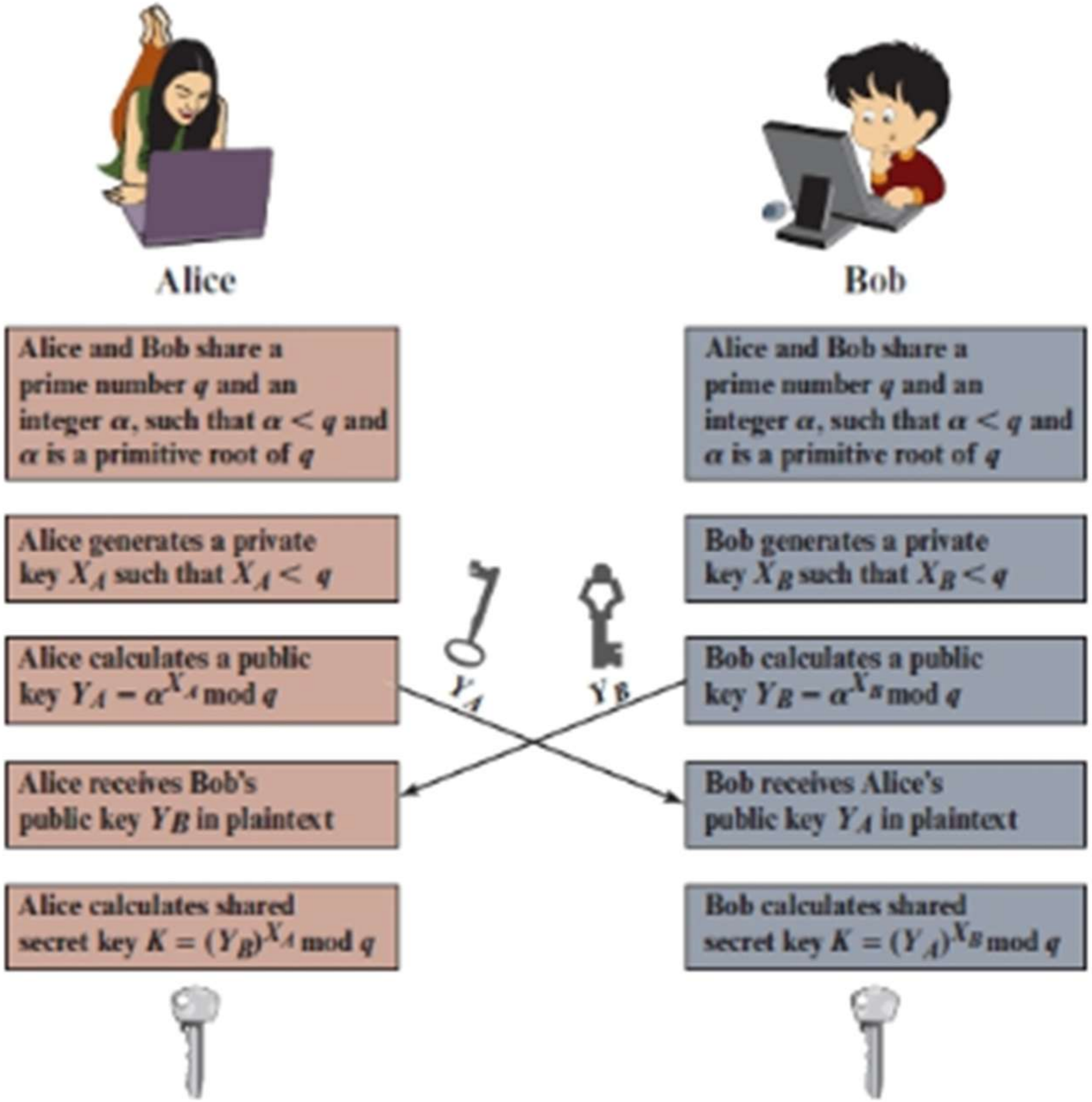
Diffie-Hellman Key Exchange

- shared session key for users A & B is KAB:
  - $K_{AB} = a^{x_A \cdot x_B} \bmod q$
  - $= Y_A^{x_B} \bmod q$ (which B can compute)
  - $= Y_B^{x_A} \bmod q$ (which A can compute)
- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the same key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

Key Exchange Protocols

- users could create random private/public D-H keys each time they communicate
- users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them
- both of these are vulnerable to a Man-in-the-Middle Attack
- authentication of the keys is needed

Key Exchange Protocols



Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q

Alice generates a private key X_A such that $X_A < q$

Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$

Alice receives Bob's public key Y_B in plaintext

Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$

Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q

Bob generates a private key X_B such that $X_B < q$

Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$

Bob receives Alice's public key Y_A in plaintext

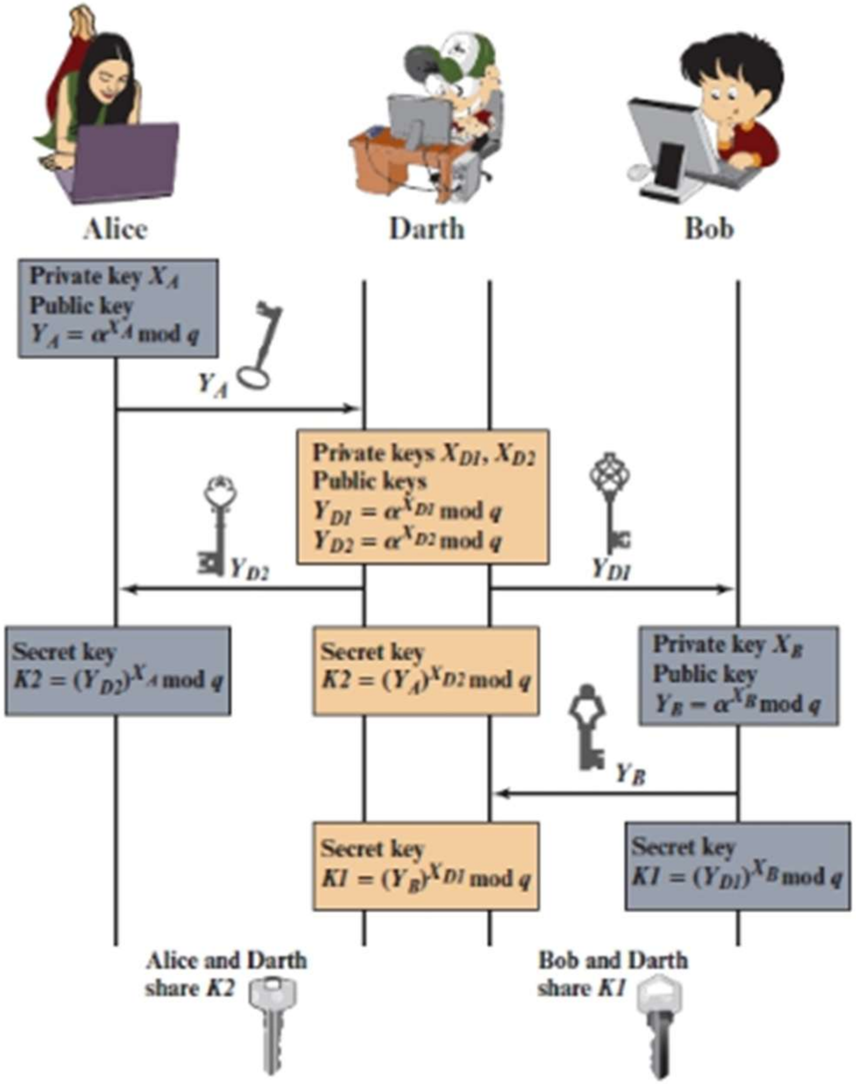
Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime q=353 and a=3
- select random secret keys:
  - A chooses x_A=97, B chooses x_B=233
- compute respective public keys:
  - $y_A=3^{97} \bmod 353 = 40$ (Alice)
  - $y_B=3^{233} \bmod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB}= y_B^{x_A} \bmod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \bmod 353 = 40^{233} = 160$ (Bob)

Man-in-the-Middle Attack

- Darth prepares by creating two private / public keys
- Alice transmits her public key to Bob
- Darth intercepts this and transmits his first public key to Bob. Darth also calculates a shared key with Alice
- Bob receives the public key and calculates the shared key (with Darth instead of Alice)
- Bob transmits his public key to Alice
- Darth intercepts this and transmits his second public key to Alice. Darth calculates a shared key with Bob
- Alice receives the key and calculates the shared key (with Darth instead of Bob)
- Darth can then intercept, decrypt, re-encrypt, forward all messages between Alice & Bob


Alice
Darth
Bob
Private key X_A
Public key
$Y_A = \alpha^{X_A} \bmod q$
Y_A
Private keys X_{D1}, X_{D2}
Public keys
$Y_{D1} = \alpha^{X_{D1}} \bmod q$
$Y_{D2} = \alpha^{X_{D2}} \bmod q$
Y_{D2}
Y_{D1}
Secret key
$K2 = (Y_{D2})^{X_A} \bmod q$
Secret key
$K2 = (Y_A)^{X_{D2}} \bmod q$
Private key X_B
Public key
$Y_B = \alpha^{X_B} \bmod q$
Y_B
Secret key
$K1 = (Y_B)^{X_{D1}} \bmod q$
Secret key
$K1 = (Y_{D1})^{X_B} \bmod q$
Alice and Darth share $K2$
Bob and Darth share $K1$

Thank you – Q&A