Chapter 7 Public Key Cryptography and Digital Signatures

Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed.

—*The Golden Bough,* Sir James George Frazer

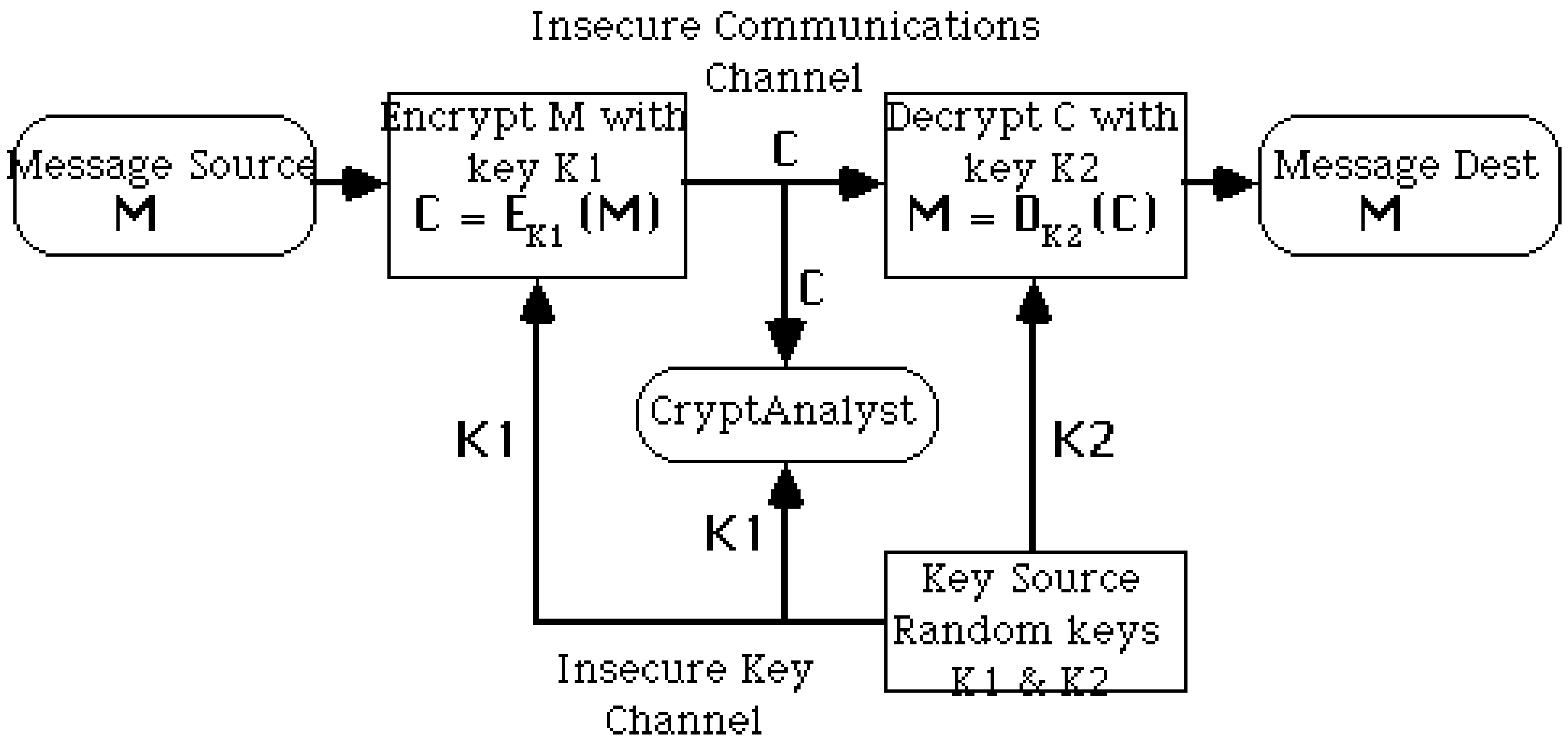
Private-Key Cryptography

- 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

Public-Key Cryptography



Asymmetric (Public-Key) Encryption System

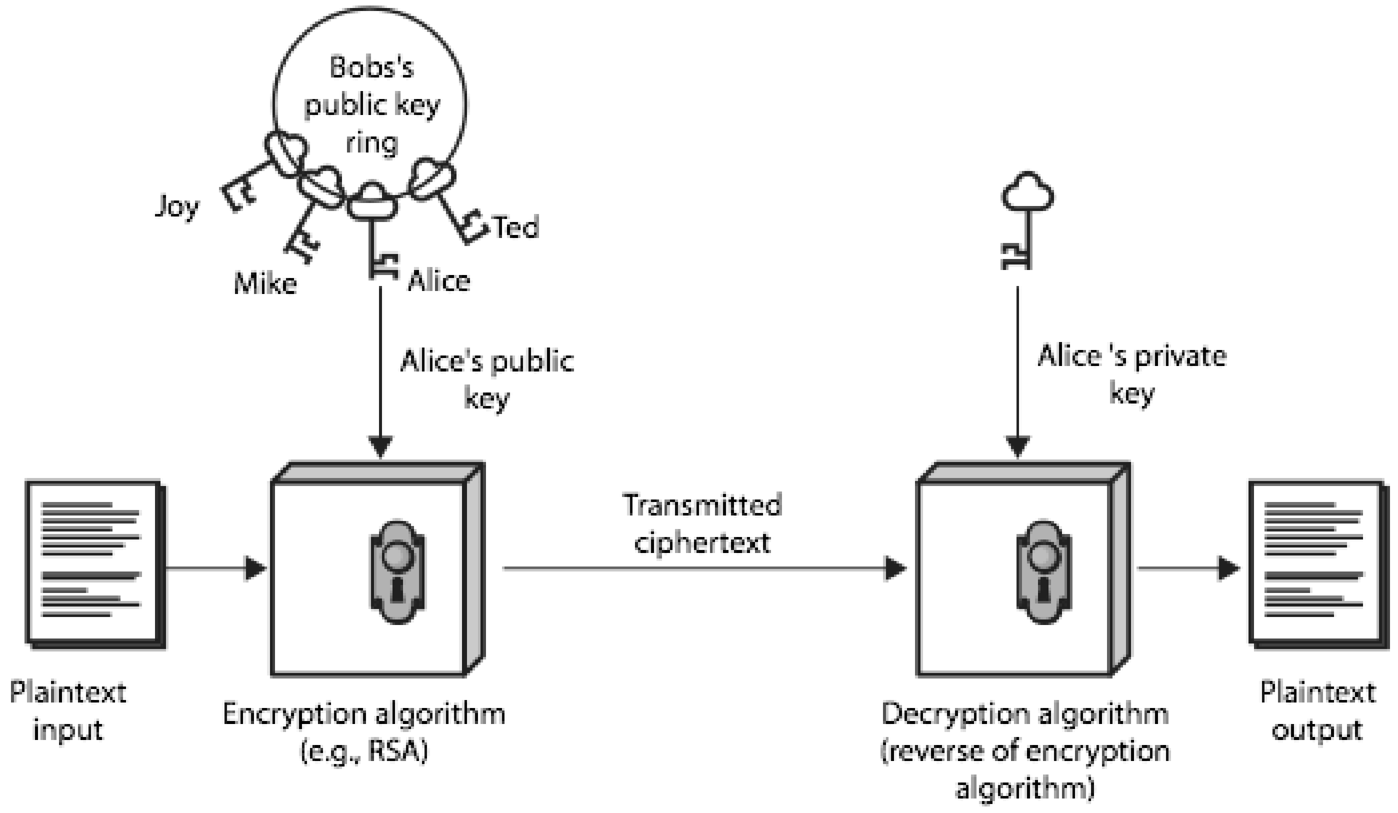
Why Public-Key Cryptography?

- 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 **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

Public-Key Cryptography

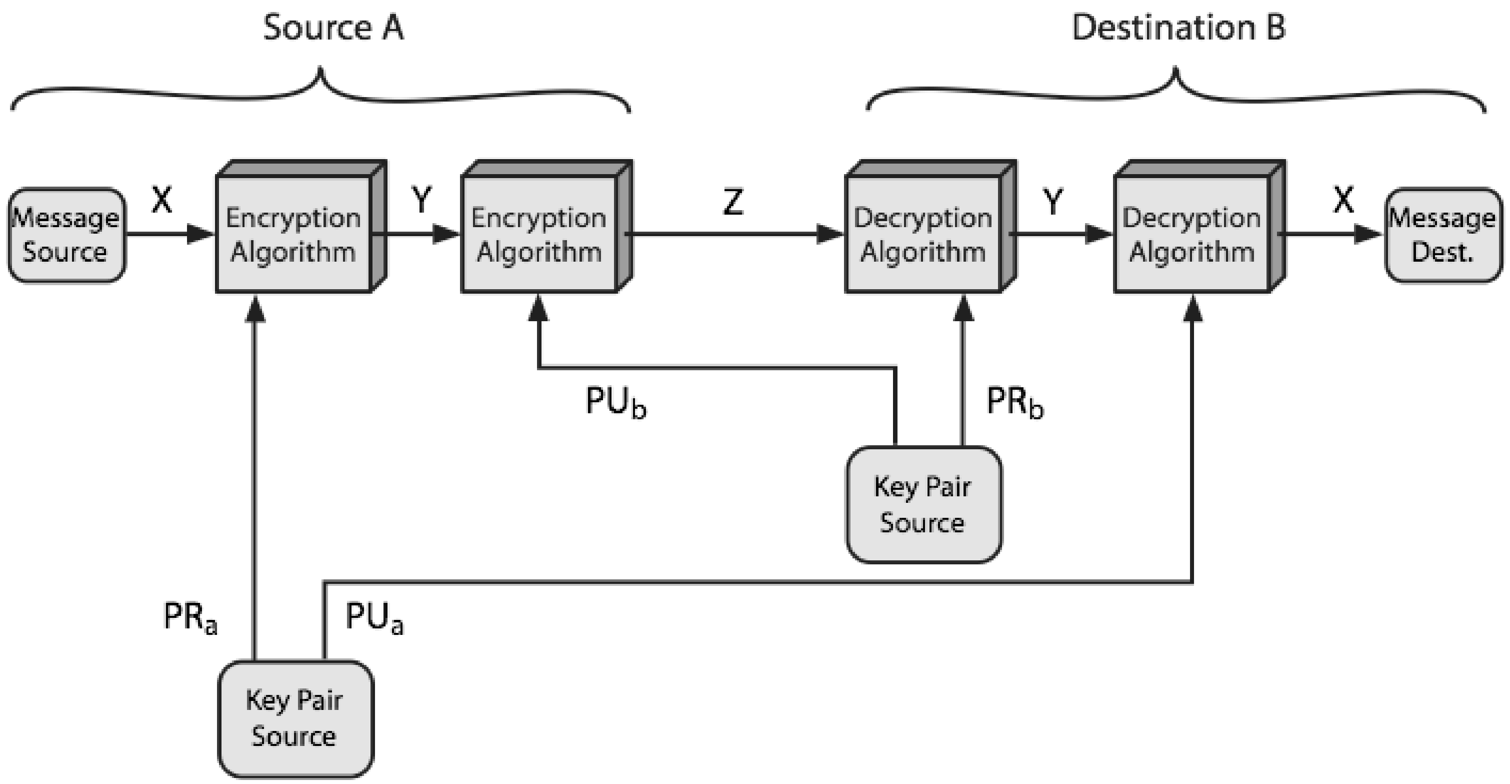


(a) Encryption

Public-Key Characteristics

- 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)

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

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

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 K_{AB}:

  $K_{AB} = a^{x_A . x_B} \bmod q$

  $= y_A^{x_B} \bmod q$ (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

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)

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 meet-in-the-Middle Attack
- authentication of the keys is needed

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 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 $\phi(n)=(p-1)(q-1)$
- selecting at random the encryption key e
  - where $1<e<\phi(n)$, $\gcd(e,\phi(n))=1$
- solve following equation to find decryption key d
  - $e.d=1 \bmod \phi(n)$ and $0\le d\le n$
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,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 $0 \le M < n$
- 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)

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. Compute $n = pq$ =17 x 11=187
3. Compute $\phi(n)=(p-1)(q-1)$=16 x 10=160
4. Select e: gcd(e,160)=1; choose e=7
5. Determine d: de=1 mod 160 and d < 160
   Value is d=23 since 23x7=161= 10x160+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 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

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)

El Gamal

a variant of the Diffie-Hellman key distribution scheme,

- published in 1985 by ElGamal in T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms“,
- like Diffie-Hellman its security depends on the difficulty of factoring logarithms
- **Key Generation** α
  - select a large prime p (~200 digit), and
  - α a primitive element mod p
  - A has a secret number x_A
  - B has a secret number x_B
  - A and B compute y_A and y_B respectively, which are then made public

    $y_A = \alpha^{xA} \bmod p$

    $y_B = \alpha^{xB} \bmod p$

El Gamal

a variant of the Diffie-Hellman key distribution scheme,

- to **encrypt** a message **M** into ciphertext **C**,
  - selects a random number **k,** $0 <= k <= p-1$
  - computes the message key **K**

    $K = y_B^k \bmod p$
  - computes the ciphertext pair: C = {c1,c2}

    $C_1 = [[alpha]]^k \bmod p$ $C_2 = K.M \bmod p$
- to **decrypt** the message
  - extracts the message key **K**

    $K = C_1^{xB} \bmod p = [[alpha]]^{k.xB} \bmod p$
  - extracts **M** by solving for M in the following equation:

    $C_2 = K.M \bmod p$

Key Management

- public-key encryption helps address key distribution problems
- have two aspects of this:
  - distribution of public keys
  - use of public-key encryption to distribute secret keys

Distribution of Public Keys

- can be considered as using one of:
  - public announcement
  - publicly available directory
  - public-key authority
  - public-key certificates

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

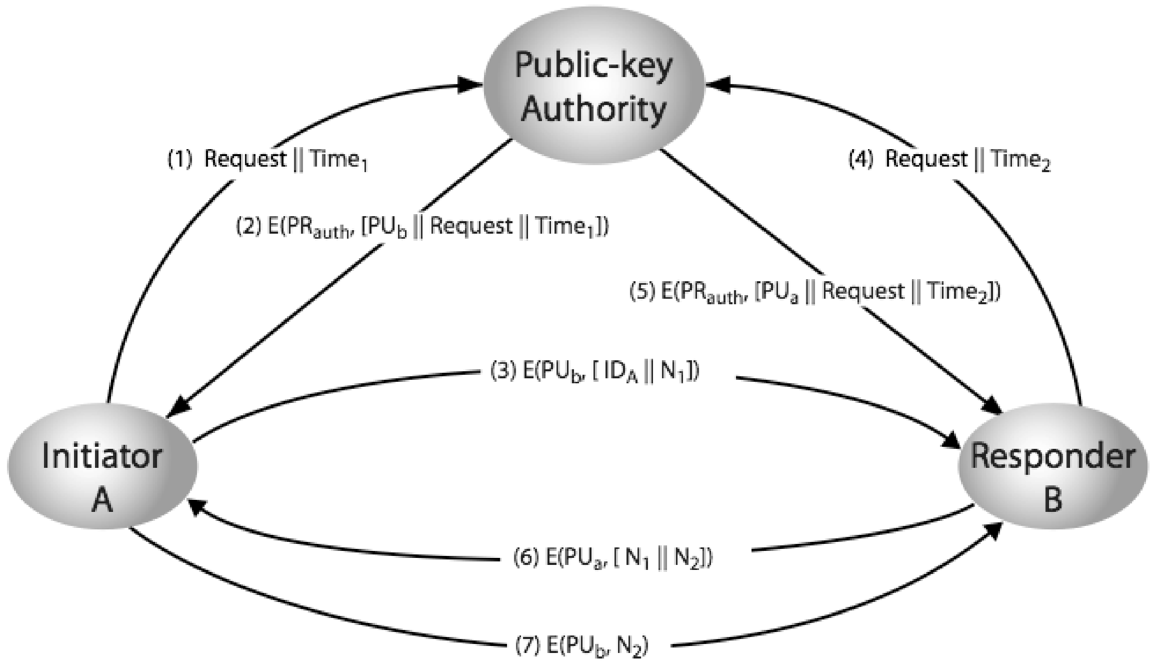
Publicly Available Directory

- 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

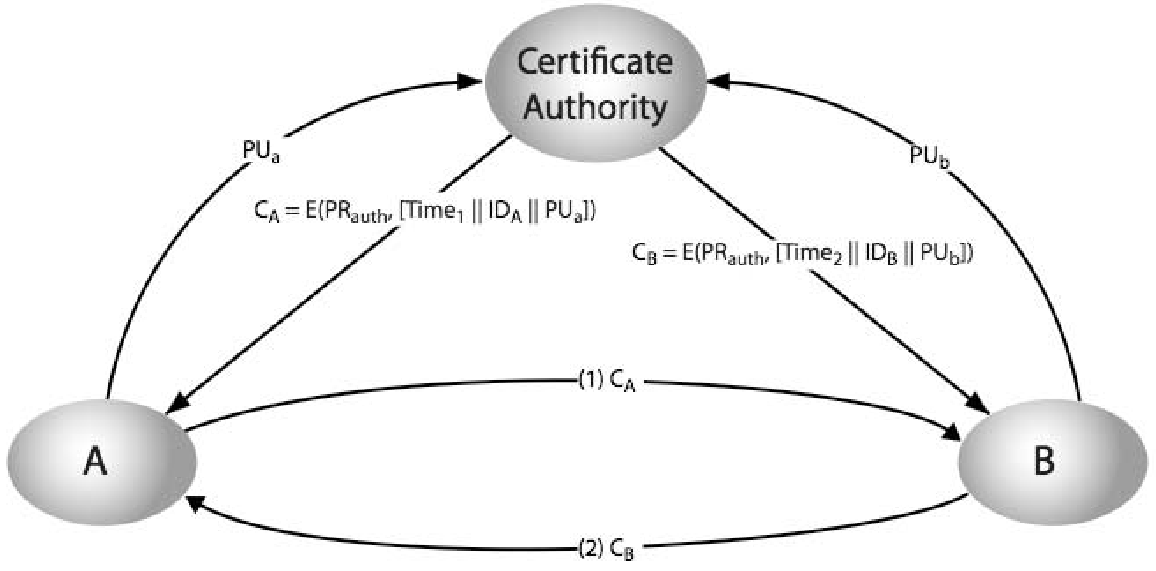
Public-Key Authority


Public-key Authority
Initiator A
Responder B
(1) Request || $Time_1$
(2) $E(PR_{auth}, [PU_b \| Request \| Time_1])$
(3) $E(PU_b, [ID_A \| N_1])$
(4) Request || $Time_2$
(5) $E(PR_{auth}, [PU_a \| Request \| Time_2])$
(6) $E(PU_a, [N_1 \| N_2])$
(7) $E(PU_b, N_2)$

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

Public-Key Certificates



Certificate Authority
PU_a
PU_b
$C_A = E(PR_{auth}, [Time_1 \| ID_A \| PU_a])$
$C_B = E(PR_{auth}, [Time_2 \| ID_B \| PU_b])$
A
B
(1) C_A
(2) C_B

Public-Key Distribution of Secret Keys

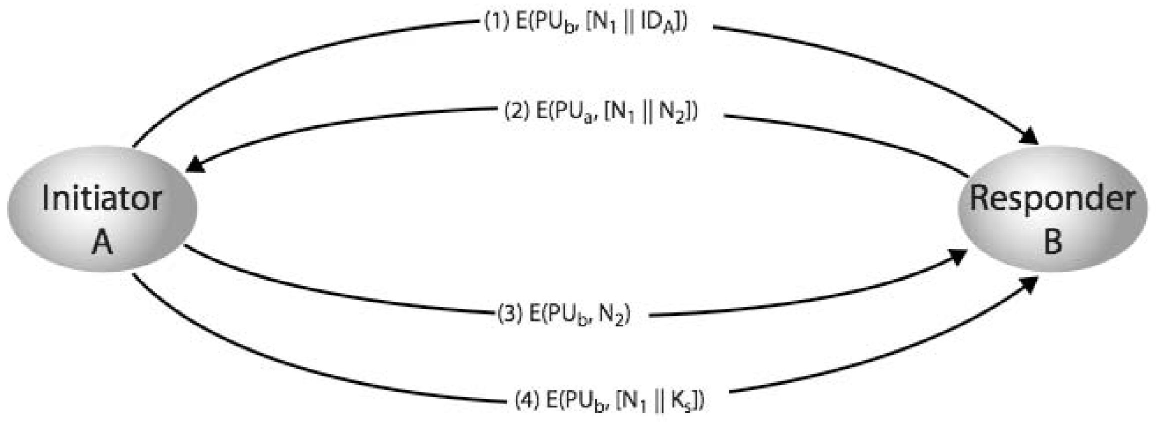
- use previous methods to obtain public-key
- can use for secrecy or authentication
- but public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session

Simple Secret Key Distribution

- proposed by Merkle in 1979
  - A generates a new temporary public key pair
  - A sends B the public key and their identity
  - B generates a session key K sends it to A encrypted using the supplied public key
  - A decrypts the session key and both use
- problem is that an opponent can intercept and impersonate both halves of protocol

Public-Key Distribution of Secret Keys

- if have securely exchanged public-keys:



(1) $E(PU_b, [N_1 || ID_A])$
(2) $E(PU_a, [N_1 || N_2])$
Initiator A
Responder B
(3) $E(PU_b, N_2)$
(4) $E(PU_b, [N_1 || K_s])$

Hybrid Key Distribution

- retain use of private-key KDC
- shares secret master key with each user
- distributes session key using master key
- public-key used to distribute master keys
  - especially useful with widely distributed users
- rationale
  - performance
  - backward compatibility

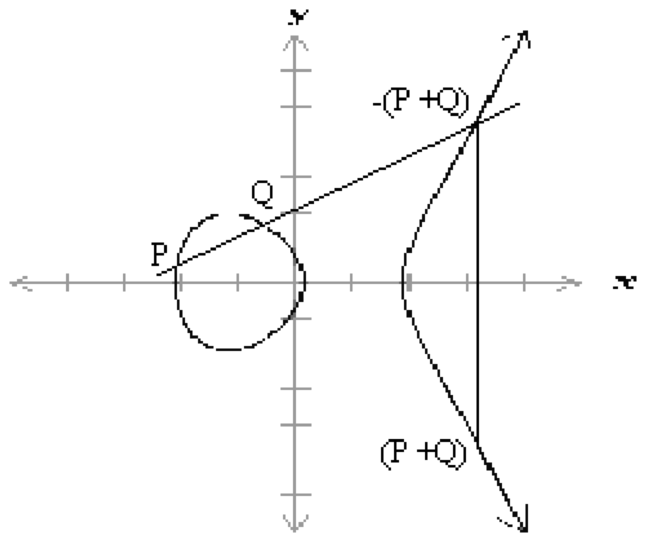
Elliptic Curve Cryptography

- majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- imposes a significant load in storing and processing keys and messages
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes
- newer, but not as well analysed

Real Elliptic Curves

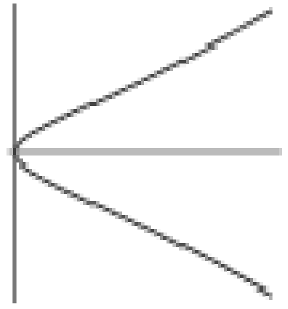
- an elliptic curve is defined by an equation in two variables x & y, with coefficients
- consider a cubic elliptic curve of form
  - $y^2 = x^3 + ax + b$
  - where x,y,a,b are all real numbers
  - if $4a^3 + 27b^2 \neq 0$ elliptic curve can be used to form group
  - also define zero point O
- have addition operation for elliptic curve
  - geometrically sum of Q+R is reflection of intersection R

Real Elliptic Curve Example

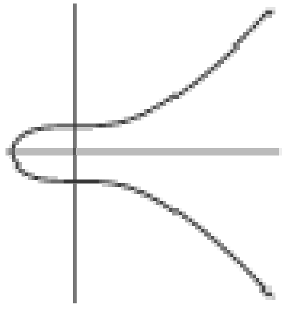


$y^2 = x^3 - 4x + 0.67$

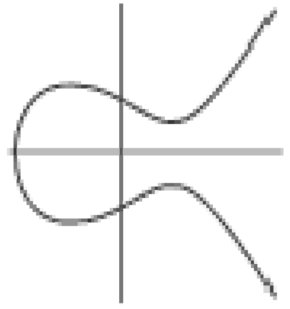
$y^2 = x^3 - 1$



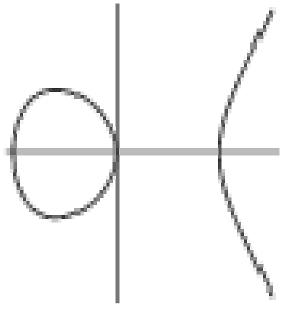
$y^2 = x^3 + 1$



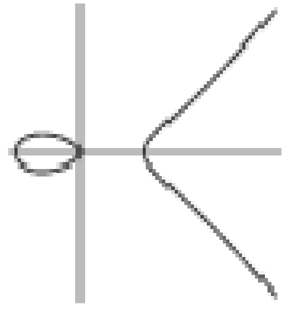
$y^2 = x^3 - 3x + 3$



$y^2 = x^3 - 4x$



$y^2 = x^3 - x$



Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime
    - best in software
  - binary curves $E_{2m}(a,b)$ defined over $GF(2^n)$
    - use polynomials with binary coefficients
    - best in hardware

Finite Elliptic Curves

- ***Adding two different P and Q points**:*
- Negative of point $P = (x_p , y_p)$ is $- P = (x_p , - y_p)$.
- Coordinate of $P + Q = R$ is computed as.
- $X_r = [\lambda^2 - x_p – x_q]$ modp
- $y_r = [- y_p + \lambda(x_p – x_r)]$ modp
- where $\lambda = (y_p – y_q)/ (x_p – x_q)$ is slope of two points.

Finite Elliptic Curves

- ***To double a point P*** = (x_p , y_p).
- $2P=R(X_r,Y_r)$
- $X_r = [\lambda^2 - 2x_p]$ modp
- $y_r = [- y_p + \lambda(x_p – x_r)]$ modp
-
- where $\lambda = (x_p{}^2 – a) / (2y_p)$ is slope and a parameter of curve equation.

Elliptic Curve Cryptography

- ECC addition is analog of modulo multiply
- ECC repeated addition is analog of modulo exponentiation
- need “hard” problem equiv to discrete log
  - `Q=kP`, where Q,P belong to a prime curve
  - is “easy” to compute Q given k,P
  - but “hard” to find k given Q,P
  - known as the elliptic curve logarithm problem
- Certicom example: $E_{23}(9,17)$

ECC Diffie-Hellman

- can do key exchange analogous to D-H
- users select a suitable curve $E_p(a,b)$
- select base point $G=(x_1,y_1)$
  - with large order n s.t. $nG=O$
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A=n_AG$, $P_B=n_BG$
- compute shared key: $K=n_AP_B$, $K=n_BP_A$
  - same since $K=n_An_BG$

ECC Encryption/Decryption

- several alternatives, will consider simplest
- must first encode any message M as a point on the elliptic curve P_m
- select suitable curve & point G as in D-H
- each user chooses private key $n_A < n$
- and computes public key $P_A = n_A G$
- to encrypt P_m : $C_m = \{kG, \quad P_m + kP_b\}$, k random
- decrypt C_m compute:

$$P_m + kP_b - n_B(kG) \;=\; P_m + k(n_B G) - n_B(kG) \;=\; P_m$$

ECC Security

- relies on elliptic curve logarithm problem
- fastest method is “Pollard rho method”
- compared to factoring, can use much smaller key sizes than with RSA etc
- for equivalent key lengths computations are roughly equivalent
- hence for similar security ECC offers significant computational advantages

Comparable Key Sizes for Equivalent Security

Symmetric scheme (key size in bits)	**ECC-based scheme (size of *n* in bits)**	**RSA/DSA (modulus size in bits)**
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

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

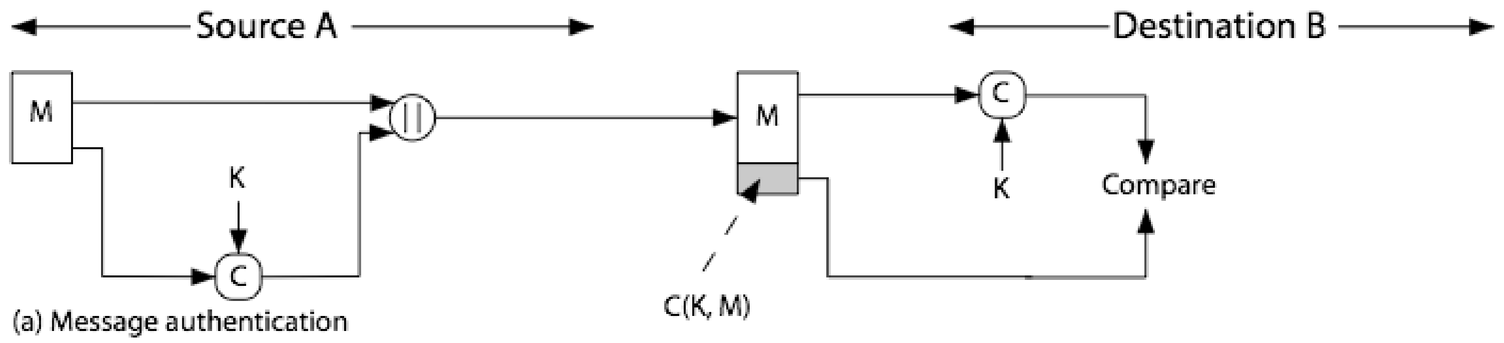
Message Encryption

- 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

  $\mathtt{MAC} = \mathtt{C}_{\mathtt{K}}(\mathtt{M})$

  – 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

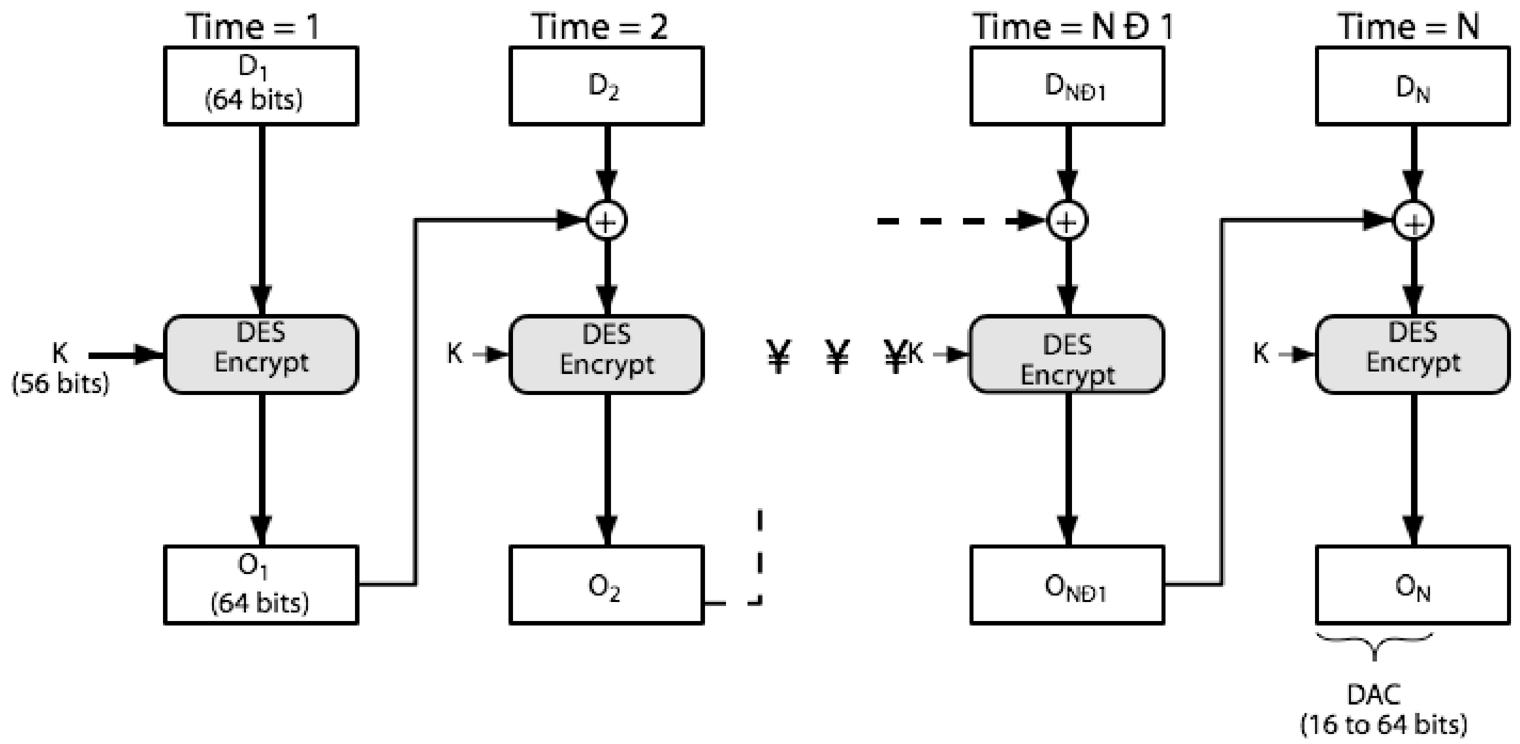
Requirements for MACs

- 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 ($16 \le M \le 64$) of final block
- but final MAC is now too small for security

Data Authentication Algorithm

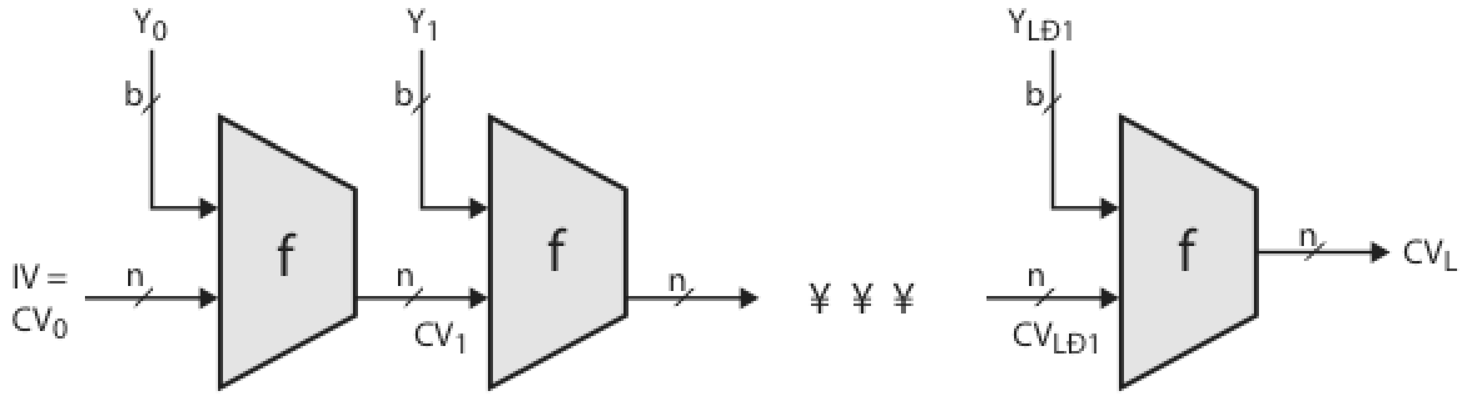


Hash Functions

- condenses arbitrary message to fixed size

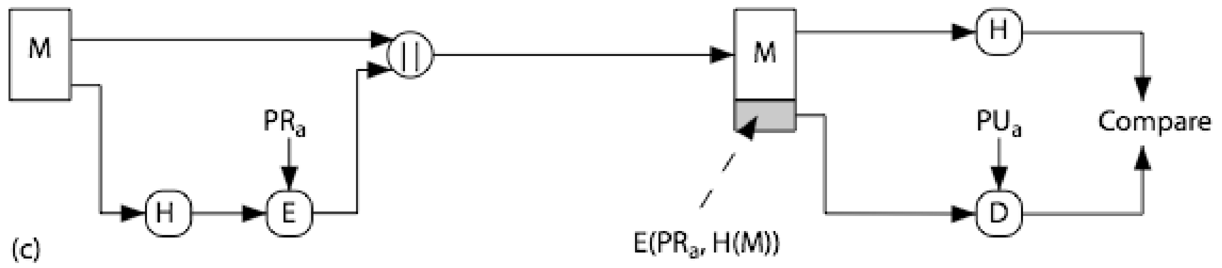
  `h = H(M)`
- usually assume that the hash function is public and not keyed
  - cf. MAC which is keyed
- hash used to detect changes to message
- can use in various ways with message
- most often to create a digital signature

General structure of Hash Functions



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

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 (next chapter)

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

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 Symmetric Encryption

- as discussed previously can use a two-level hierarchy of keys
- usually with a trusted Key Distribution Center (KDC)
  - each party shares own master key with KDC
  - KDC generates session keys used for connections between parties
  - master keys used to distribute these to them

Needham-Schroeder Protocol

- original third-party key distribution protocol
- for session between A B mediated by KDC
- protocol overview is:
  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: $E_{Kb}[Ks||ID_A]$
  4. B -> A: $E_{Ks}[N_2]$
  5. A -> B: $E_{Ks}[f(N_2)]$

Needham-Schroeder Protocol

- used to securely distribute a new session key for communications between A & B
- but is vulnerable to a replay attack if an old session key has been compromised
  - then message 3 can be resent convincing B that is communicating with A
- modifications to address this require:
  - timestamps (Denning 81)
  - using an extra nonce (Neuman 93)

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

Denning AS Protocol

- Denning 81 presented the following:
  1. A -> AS: $ID_A \,||\, ID_B$
  2. AS -> A: $E_{PRas}[ID_A||PU_a||T] \,||\, E_{PRas}[ID_B||PU_b||T]$
  3. A -> B: $E_{PRas}[ID_A||PU_a||T] \,||\, E_{PRas}[ID_B||PU_b||T] \,||\, E_{PUb}[E_{PRas}[K_s||T]]$
- note session key is chosen by A, hence AS need not be trusted to protect it
- timestamps prevent replay but require synchronized clocks

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

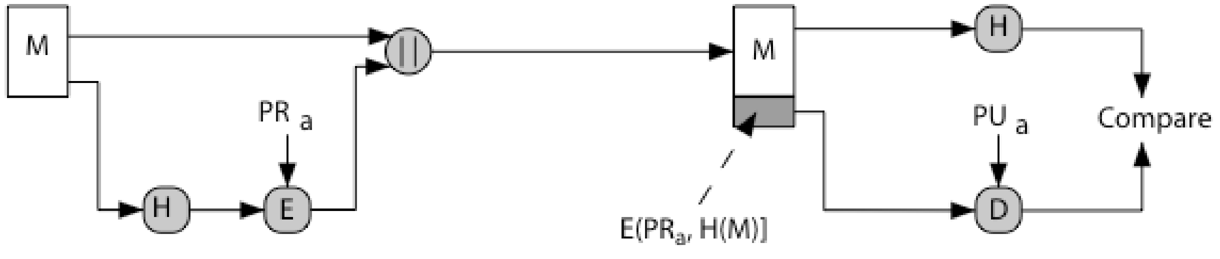
Digital Signature Standard (DSS)

- 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

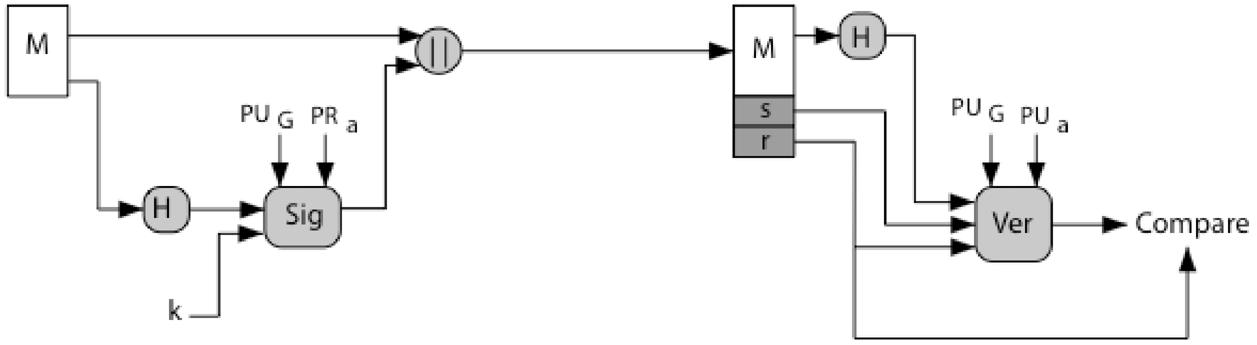
Digital Signature Algorithm (DSA)

- 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

Digital Signature Algorithm (DSA)



(a) RSA Approach



(b) DSS Approach

DSA Key Generation

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

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:

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

  $s = (k^{-1}.H(M) + x.r) \pmod q$
- sends signature (r,s) with message M

DSA Signature Verification

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

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

$u1 = (H(M).w) \pmod q$

$u2 = (r.w) \pmod q$

$v = (g^{u1}.y^{u2} \pmod p) \pmod q$

- if `v=r` then signature is verified
- see book web site for details of proof why