Public Key Cryptography and Cryptographic Hashes

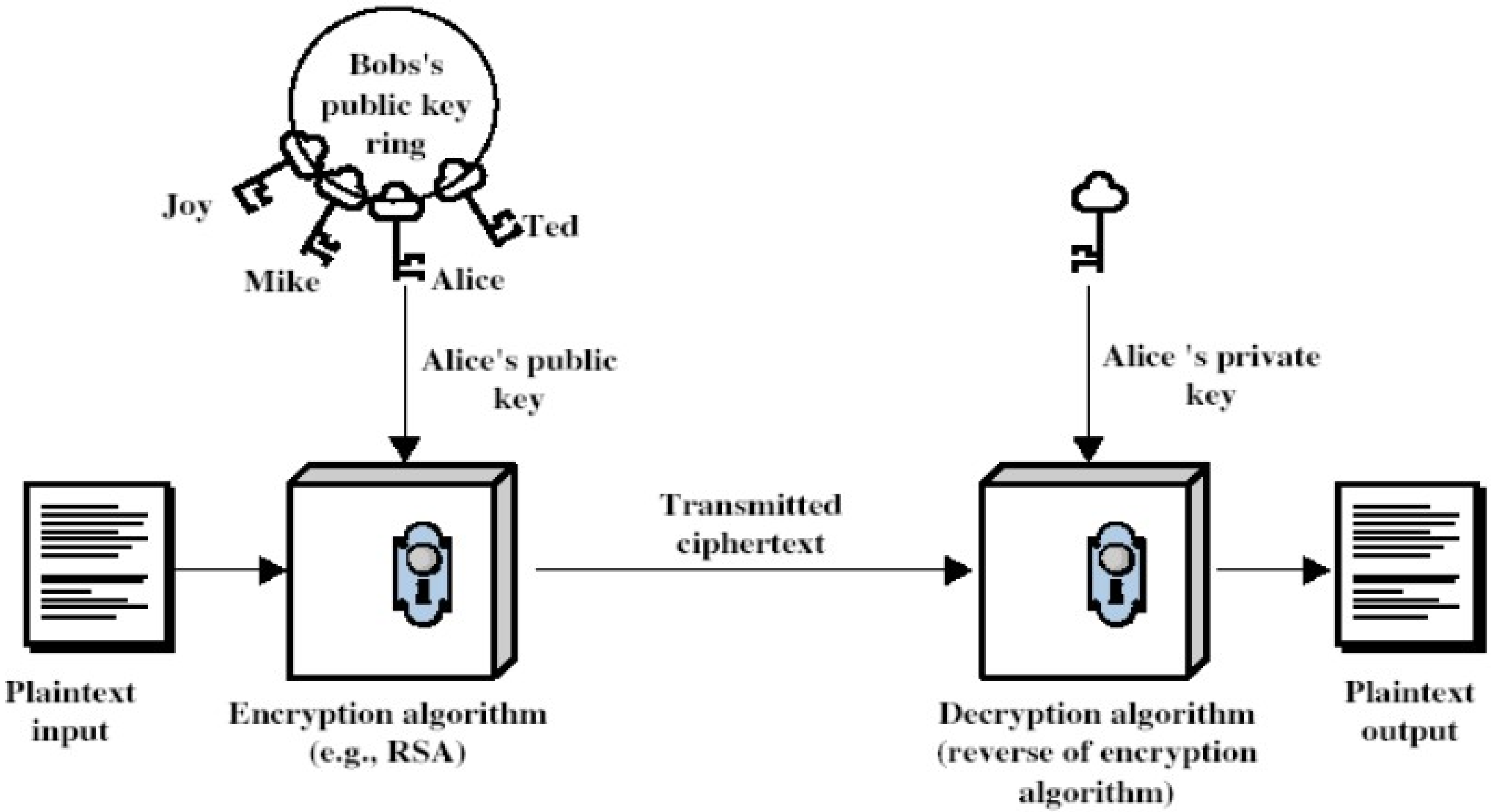
CS461/ECE422

Fall 2009

Reading

- Computer Security: Art and Science – Chapter 9
- Handbook of Applied Cryptography, Chapter 8
  - http://www.cacr.math.uwaterloo.ca/hac/

Public-Key Cryptography



Public Key Cryptography

- Two keys
  - *Private key* known only to individual
  - *Public key* available to anyone
- Idea
  - Confidentiality: encipher using public key, decipher using private key
  - Integrity/authentication: encipher using private key, decipher using public one

Requirements

1. It must be computationally easy to encipher or decipher a message given the appropriate key
2. It must be computationally infeasible to derive the private key from the public key
3. It must be computationally infeasible to determine the private key from a chosen plaintext attack

General Facts about Public Key Systems

- Public Key Systems are much slower than Symmetric Key Systems
  - RSA 100 to 1000 times slower than DES. 10,000 times slower than AES?
  - Generally used in conjunction with a symmetric system for bulk encryption
- Public Key Systems are based on “hard” problems
  - Factoring large composites of primes, discrete logarithms, elliptic curves
- Only a handful of public key systems perform both encryption and signatures

Diffie-Hellman

- The first public key cryptosystem proposed
- Usually used for exchanging keys securely
- Compute a common, shared key
  - Called a *symmetric key exchange protocol*
- Based on discrete logarithm problem
  - Given integers n and g and prime number p, compute k such that $n = g^k \bmod p$
  - Solutions known for small p
  - Solutions computationally infeasible as p grows large

Algorithm

- Public Constants: prime p, integer $g \neq 0, 1,$ or $p-1$
- Choose private keys and compute public keys
  - Anne chooses private key *kAnne*, computes public key $KAnne = g^{kAnne} \bmod p$
  - Similarly Bob chooses *kBob*, computes $Kbob = \mathrm{g}^{kBob} \bmod \mathrm{p}$
- Exchange public keys and compute shared information
  - To communicate with Bob, Anne computes $Kshared = KBob^{kAnne} \bmod p$
  - To communicate with Anne, Bob computes $Kshared = KAnne^{kBob} \bmod p$

Working the Equations

- $(KBob)^{kAnne}$ mod p
- $= (g^{kBob}$ mod p$)^{kAnne}$ mod p
- $= g^{kBob\ kAnne}$ mod p

- $(Kalice)^{kBob}$ mod p
- $= (g^{kAlice}$ mod p$)^{kBob}$ mod p
- $= g^{kAlice\ kBob}$ mod p

- If Eve sees Kalice and Kbob, why can't she compute the common key?

Example

- Assume $p = 53$ and $g = 17$
- Alice chooses $kAlice = 5$
  - Then $KAlice = 17^5 \bmod 53 = 40$
- Bob chooses $kBob = 7$
  - Then $KBob = 17^7 \bmod 53 = 6$
- Shared key:
  - $KBob^{kAlice} \bmod p = 6^5 \bmod 53 = 38$
  - $KAlice^{kBob} \bmod p = 40^7 \bmod 53 = 38$

Real public DH values

- For IPSec and SSL, there are a small set of g's and p's published that all standard implementations support.
  - Group 1 and 2
    - http://tools.ietf.org/html/rfc2409
  - Group 5 and newer proposed values
    - http://tools.ietf.org/html/draft-ietf-ipsec-ike-modp-gro

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)

Modular Arithmetic

- a mod b = x if for some k >= 0, bk + x = a
- Associativity, Commutativity, and Distributivity hold in Modular Arithmetic
- Inverses also exist in modular arithmetic
  - $a + (-a) \bmod n = 0$
  - $a * a^{-1} \bmod n = 1$

Modular Arithmetic

- Reducibility also holds
  - (a + b) mod n = (a mod n + b mod n) mod n
  - a * b mod n = ((a mod n) * b mod n) mod n
- Fermat's Thm: if p is any prime integer and a is an integer, then a^p mod p = a
  - Corollary: a^{p-1} mod p = 1 if a != 0 and a is relatively prime to p

Background

- Totient function $\phi(n)$
  - Number of positive integers less than *n* and relatively prime to *n*
    - *Relatively prime* means with no factors in common with *n*
- Example: $\phi(10) = ?$
  - 4 because 1, 3, 7, 9 are relatively prime to 10
- Example: $\phi(p) = ?$ where p is a prime
  - p-1 because all lower numbers are relatively prime

Background

- Euler generalized Fermat's Thm for composite numbers.
  - Recall Fermat's Thm $a^{p-1}=1 \bmod p$ if a != 0

- Euler's Thm: $x^{\phi(n)}=1 \bmod n$
  - Where q and p are primes
  - $n = pq$
  - then $\phi(n) = (p-1)(q-1)$

RSA Algorithm

- Choose two large prime numbers p, q
  - Let $n = pq$; then $\phi(n) = (p-1)(q-1)$
  - Choose $e < n$ such that e is relatively prime to $\phi(n)$.
  - Compute d such that $ed \bmod \phi(n) = 1$
- Public key: (e, n); private key: d
- Encipher: $c = m^e \bmod n$
- Decipher: $m = c^d \bmod n$
- Generically: $\mathrm{F(V, x) = V^x \bmod n}$

Working through the equations

- $C = F(M, e) = M^e \bmod n$
- $M = F(F(M, e), d)$
- $M = (M^e \bmod n)^d \bmod n$
- $M = M^{ed} \bmod n$
  - $ed \bmod \phi(n) = 1$
  - $k * \phi(n) + 1 = ed$
- $M = (M \bmod n * M^{k\,\phi(n)} \bmod n) \bmod n$
  - By Euler' theorem $X^{\phi(n)} \bmod n = 1$
- $M = M \bmod n$

Where is the security?

- What problem must you solve to discover d?
- Public key: (e, n); private key: d

Security Services

- Confidentiality
  – Only the owner of the private key knows it, so text enciphered with public key cannot be read by anyone except the owner of the private key
- Authentication
  – Only the owner of the private key knows it, so text enciphered with private key must have been generated by the owner

More Security Services

- Integrity
  – Enciphered letters cannot be changed undetectably without knowing private key
- Non-Repudiation
  – Message enciphered with private key came from someone who knew it

Example: Confidentiality

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
- Bob wants to send Alice secret message HELLO (07 04 11 11 14)
  - $07^{17} \bmod 77 = 28$
  - $04^{17} \bmod 77 = 16$
  - $11^{17} \bmod 77 = 44$
  - $11^{17} \bmod 77 = 44$
  - $14^{17} \bmod 77 = 42$
- Bob sends 28 16 44 44 42

Example

- Alice receives 28 16 44 44 42
- Alice uses private key, d = 53, to decrypt message:
  - $28^{53} \bmod 77 = 07$
  - $16^{53} \bmod 77 = 04$
  - $44^{53} \bmod 77 = 11$
  - $44^{53} \bmod 77 = 11$
  - $42^{53} \bmod 77 = 14$
- Alice translates message to letters to read HELLO
  - No one else could read it, as only Alice knows her private key and that is needed for decryption

Example: Integrity/Authentication

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
- Alice wants to send Bob message HELLO (07 04 11 11 14) so Bob knows it is what Alice sent (no changes in transit, and authenticated)
  - $07^{53} \bmod 77 = 35$
  - $04^{53} \bmod 77 = 09$
  - $11^{53} \bmod 77 = 44$
  - $11^{53} \bmod 77 = 44$
  - $14^{53} \bmod 77 = 49$
- Alice sends 35 09 44 44 49

Example

- Bob receives 35 09 44 44 49
- Bob uses Alice's public key, $e = 17$, $n = 77$, to decrypt message:
  - $35^{17} \bmod 77 = 07$
  - $09^{17} \bmod 77 = 04$
  - $44^{17} \bmod 77 = 11$
  - $44^{17} \bmod 77 = 11$
  - $49^{17} \bmod 77 = 14$
- Bob translates message to letters to read HELLO
  - Alice sent it as only she knows her private key, so no one else could have enciphered it
  - If (enciphered) message's blocks (letters) altered in transit, would not decrypt properly

Example: Both

- Alice wants to send Bob message HELLO both enciphered and authenticated (integrity-checked)
  - Alice's keys: public (17, 77); private: 53
  - Bob's keys: public: (37, 77); private: 13
- Alice enciphers HELLO (07 04 11 11 14):
  - $(07^{53} \bmod 77)^{37} \bmod 77 = 07$
  - $(04^{53} \bmod 77)^{37} \bmod 77 = 37$
  - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
  - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
  - $(14^{53} \bmod 77)^{37} \bmod 77 = 14$
- Alice sends 07 37 44 44 14

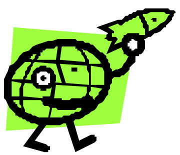
Warnings

- Encipher message in blocks considerably larger than the examples here
  - If 1 character per block, RSA can be broken using statistical attacks (just like classical cryptosystems)
  - Attacker cannot alter letters, but can rearrange them and alter message meaning
    - Example: reverse enciphered message of text ON to get NO

Direct Digital Signature

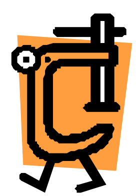
- Involve only sender & receiver
- Assumed receiver has sender’s public-key
- Digital signature made by sender signing entire message or hash with private-key
- Can encrypt using receivers public-key
- Important that sign first then encrypt message & signature
- Security depends on sender’s private-key

Potential problems



Alice

Bob



Carol

Sign-Encrypt vs. Encrypt-Sign

- Is Sign-Encrypt Enough?
  - Recipient knows who wrote the message
  - But who encrypted it?
  - Surreptitious forwarding
- Does Encrypt-Sign make sense?
  - Signature can be easily replaced
  - RSA Signatures

Options to Fix

- Naming repairs
  - Include Senders name
  - Include Recipients name
- Sign/Encrypt/Sign
- Encrypt/Sign/Encrypt
- Which is the best?
  - Add recipient’s name, Sign and Encrypt
  - Other solutions all require extra hash (of message or key)

Hash or Checksums

- Mathematical function to generate a set of k bits from a set of n bits (where $k \leq n$).
  - k is smaller then n except in unusual circumstances
- Example: ASCII parity bit
  - ASCII has 7 bits; 8th bit is "parity"
  - Even parity: even number of 1 bits
  - Odd parity: odd number of 1 bits

Example Use

- Bob receives “10111101” as bits.
  - Sender is using even parity; 6 1 bits, so character was received correctly
    - Note: could be garbled, but 2 bits would need to have been changed to preserve parity
  - Sender is using odd parity; even number of 1 bits, so character was not received correctly

Another Example

- 8-bit Cyclic Redundancy Check (CRC)
  - XOR all bytes in the file/message
  - Good for detecting accidental errors
  - But easy for malicious user to "fix up" to match altered message
- For example, change the 4^{th} bit in one of the bytes
  - Fix up by flipping the 4^{th} bit in the CRC
- Easy to find a M' that has the same CRC

Cryptographic Hash or Checksum or Message Digest

- $h: A \rightarrow B$:
  - For any $x \in A$, $h(x)$ is easy to compute
  - For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$
    - One way function, e.g., computing x^3 vs cube root of x by hand
  - It is computationally infeasible to find two inputs x, $x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$
    - Alternate form: Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$.

Collisions

- If $x \neq x'$ and $h(x) = h(x')$, x and x' are a *collision*
  - Pigeonhole principle: if there are n containers for $n+1$ objects, then at least one container will have 2 objects in it.
  - Application: if there are 32 files and 8 possible cryptographic checksum values, at least one value corresponds to at least 4 files
  - How many files until you are guaranteed a collision? Until you probably have a collision?

Birthday Paradox

- What is the probability that someone in the room has the same birthday as me?
- What is the probability that two people in the room have the same birthday?
  - $P(n) = 1 - (365!/(365^n*(365-n)!)$
  - $P(n) > ½$ for $n = 23$
  - Section 2.15 – Handbook of Applied Cryptography
  - http://en.wikipedia.org/wiki/Birthday_paradox

Birthday Paradox

- In general, probability of a collision of reaches 50% for *M* units when
  - $n = sqrt(M)$
- If hash has m bits, this means $M = 2^m$ possible hash values
  - $n = 2^{m/2}$ for 50% probability collision

Another View of Collisions

- **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
- Need to use larger MACs

Keyed vs Keyless Hash

- Keyless hash – anyone can recompute given the message and indication of the algorithm
- Keyed hash – must have access to a key to recompute the hash
- When is each option appropriate?

MD5 and SHA

- Most widely used keyless crypto hashes
- Both are round based bit operations
  - Similar in spirit to AES and DES
  - Looking for avalanche effect to make output appear random
- MD5 is 128 bits and SHA-1 is 160 bits

More on SHA

- Standard put forth by NIST
- SHA spec
  - http://csrc.nist.gov/CryptoToolkit/tkhash.html
- Comes in different flavors that vary based on output size
  - SHA-1 outputs 160 bits
  - The other SHA-X flavors output X bits

SHA-1 Broken

- Chinese researchers had a break through
  - http://www.schneier.com/blog/archives/2005/02/s
  - –Recent results show that you can find collisions in 2^69 attempts which would be less than 2^80 from brute force
  - –Does not affect HMAC-SHA
- NIST published standards promoting using of larger SHA's
  - –http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html
  - –Preparing for public competition for new algorithm

Message Authentication Codes

- MAC is a crypto hash that is a proof of a message's integrity
  - Important that adversary cannot fixup MAC if he changes message
- MAC's rely on keys to ensure integrity
  - Either Crypto Hash is encrypted already
  - Or Crypto Hash must be augmented to take a key

Use Symmetric Ciphers for Keyed Hash

- Can use DES or AES in CBC mode
  - Last block is the hash
- DES with 64 bit block size is too small to be effective MAC

HMAC

- Make keyed cryptographic checksums from keyless cryptographic checksums
- h keyless cryptographic checksum function that takes data in blocks of b bytes and outputs blocks of l bytes. k' is cryptographic key of length b bytes
- *ipad* is 00110110 repeated b times
- *opad* is 01011100 repeated b times
- HMAC-$h(k, m) = h(k' \oplus opad \,||\, h(k' \oplus ipad \,||\, m))$

  $\oplus$ exclusive or, $||$ concatenation

Key Points

- Symmetric cryptosystems encipher and decipher using the same key
  - Or one key is easily derived from the other
- Public key cryptosystems encipher and decipher using different keys
  - Computationally infeasible to derive one from the other
- Cryptographic checksums provide a check on integrity