CSE 589 -- Lecture 8

Just because you're paranoid doesn't mean they're not out to get you.

Woody Allen

Plan for Today

- crypto review
- RSA
- digital signatures
- one-way hash functions
- authenticated key-exchange
- MACs
- Example: SSL

Security Risks of Internet Communication

- Eavesdropping
  - intermediaries listen in on private conversations
  - Solution: encryption (public or private-key)
- Manipulation
  - intermediaries change information in a private communication
  - Solution: methods for preserving data integrity (one-way hash fn's and MACs)
- Impersonation
  - a sender or receiver communicates under false identification
  - Solution: authentication (signatures, etc.)

Terminology

- A sender wants to send a message to a receiver securely -- wants to make sure no eavesdropper can read the message.

ciphertext

Plaintext ----> Encryption ----------> Decryption ----> Plaintext

M **E(M)** **C=E(M)** **D(C)** **M=D(C)**

plaintext -- original message

encryption -- process of disguising message so as to hide its substance

ciphertext -- encrypted message

decryption -- process of turning ciphertext back into plaintext.

cryptography -- art and science of keeping messages secure

cryptanalysis -- art and science of breaking ciphertext

The cast of characters

- Alice: first participant in all protocols
- Bob: second participant in all protocols
- Eve: eavesdropper
- Mallory: malicious active attacker
- Trent: trusted arbitrator

Encryption: Symmetric (private key) cryptography

- Alice and Bob share a key K the adversary doesn't know.
  - stream ciphers -- operate on plaintext a bit (byte) at a time
  - block ciphers -- operate on group (block) of bits.
- Examples: DES, RC4, IDEA, Blowfish....
- Notable feature: fast

Protocol for communicating using symmetric cryptography

- Alice and Bob agree on a cryptosystem
- Alice and Bob agree on a key
- Alice encrypts plaintext using encryption algorithm and key => ciphertext
- Alice sends ciphertext to Bob
- Bob decrypts ciphertext with same algorithm and key and reads it.

Problems

- keys must be distributed in secret
- if key compromised, all is lost
- # keys needed grows like Omega (n^2)

Public key algorithms:

- encryption key = public key
- decryption key = private key (secret key)

Private key cannot be calculated from public key in reasonable amount of time.

Protocol for communicating using public-key cryptography

- Alice and Bob agree on a public key- cryptosystem
- Bob sends Alice his public key, or else she finds it in database.
- Alice encrypts plaintext using Bob's public key => ciphertext
- Alice sends ciphertext to Bob
- Bob decrypts ciphertext using his private key.

Public-Key Cryptography

- Idea due to Diffie & Hellman 1976 (indep Merkle)
- First realization 1977: Rivest, Shamir, Adleman.
- Since then, many others.
- Security rests on different computational problems.
  - RSA -- factoring large integers (??)
  - McEliecee -- decoding linear code (NP-complete)
  - El Gamal -- discrete logarithm problem (??)
  - Chor-Rivest -- knapsack (NP-complete)
  - ….
- **Public-key cryptosystem can never provide unconditional security.**

Idea: uses trapdoor one-way function.

E_K should be easy to compute

D_K (inverting E_K) should be hard

=> E_K should be a **one-way** function

- Example of possible one-way function:

  n=pq (p,q two large prime numbers)

  $f(x) = x^b \bmod n$

- Don't want E_K to be one-way from Bob's point of view => Bob must possess trapdoor, secret information that permits easy inversion of E_K

One-Way Functions

- A function f is one-way if, given x it is easy to compute f(x), but given f(x) it is hard to compute x (superpolynomial, or exponential time).
- **Trapdoor one-way functions**: easy to compute f(x) given x, hard to compute x given f(x). But there is some secret information y, s.t. given f(x) and y, easy to compute x.

Review of Number-theoretic Preliminaries

- Modular Arithmetic:
  - Speed up calculations by reducing modulo n
  - Exponentiation is fast.
- Two numbers relatively prime when share no common factors.
- $a^{-1} \pmod n$ is the number x such that $ax \pmod n = 1$.
  - has unique solution if a and n are relatively prime.
  - has no solution otherwise.
- Extended Euclidean Algorithm
  - to calculate gcd(a,b)
  - to calculate $a^{-1} \pmod n$

More NT Review:

- Fermat's Little Theorem: If m is a prime and a is not a multiple of m, then
  $a^{m-1} = 1 \bmod m$
- Euler phi function $\phi(n)$ = number of positive integers less than n that are relatively prime to n (n > 1).
- If n = pq, where p and q are prime, $\phi(n) = (p-1)(q-1)$
- Euler's generalization of FLT: If gcd(a,n) = 1, then
  $a^{\phi(n)} \bmod n = 1$

More NT Review:

- Special Case of Chinese Remainder Theorem:
  p, q prime, n= pq
  There is a unique x < pq such that
  x = a mod p
  x = b mod q
- **Easy corollary:** x = a mod p, x = a mod q
  => x = a mod n.

The RSA Cryptosystem

RSA

- Bob selects at random 2 large prime numbers p and q, say 150 decimals each.
- Compute n = pq
- Bob chooses odd integer e relatively prime to $\phi(n) = (p-1)(q-1)$.
- Compute d as multiplicative inverse of e, modulo $\phi(n)$.
- Publish P=(e,n) as public key.
- Keep secret S=(d,n) as private (secret) key.

Domain of plaintexts is $\mathbf{Z}_n$

- Encryption function $E(M) = C = M^e \pmod n$
- Decryption function $D(C) = C^d \pmod n$

Example

- p=47, q = 71 => n= pq = 3337
- encryption key e must have no factors in common with (p-1)(q-1)=46 * 70 = 3220.
- Choose e at random to be 79.
  => d= $79^{-1} \bmod 3220 = 1019$. [(1019 x 79) mod 3220 = 1]
- Publish e, n; keep d secret.

- To encrypt M = 688, then E(M) = $688^{79} \bmod 3337 = 1570$
- To decrypt C= 1570, then D(C) = $1570^{1019} \bmod 3337 = 688$

Issues

- Why does it work?
- How do we implement it?
- What kind of security guarantees does it provide?

Why does it work? Encryption & decryption inverses

- N=pq, $de = 1 \bmod \phi(n)$
- $E(M) = M^e \bmod n = C$
- $D(C) = C^d \bmod n$.

- We'll show it when M and n are relatively prime

$de = 1 \bmod \phi(n)$ => $de = t\,\phi(n) + 1$ for t integer

=> $(M^e)^d \bmod n = M^{ed} \bmod n$

$= M^{t\phi(n)+1} \bmod n$

$= (M^{\phi(n)})^t M \bmod n$

$= 1^t M \bmod n$ by F.L.T.

$= M \bmod n$

Implementing RSA

- Bob generates two large primes, p & q
  - Probabilistic primality testing $O((\log n)^3)$
- Bob computes $n = pq$ and $\phi(n) = (p-1)(q-1)$
- Bob chooses random e $(1 < e < \phi(n))$ such that $\gcd(e, \phi(n)) = 1$
  - Euclidean algorithm
- Bob computes $d = e^{-1} \bmod \phi(n)$
  - Extended Euclidean Algorithm $O((\log n)^2)$
- Bob publishes n and e in a directory as his public key.

Probabilistic Primality Testing

FLT: $a^{m-1} = 1 \bmod m$, for m prime (*)

For most composite numbers, equation false for more than half the a's.

Gives way to test a number m to see if it's prime.

Choose a random $1 <= a <= m-1$.

Raise it to power m-1 to see if equation (*) is true.

If not, m isn't prime.

If is, repeat with bunch more random a's.

Probabilistic Primality Testing

- To find a random 100 digit prime number, pick random 100 digit number, and perform test.
- Keep going until you choose a number that turns out to be prime.
- Luckily, primes are plentiful.
- (Prime Number Theorem: # primes <= N about N/ ln N)
- => About 1/230 100 digit numbers are prime.

A few obvious questions….

- If everyone needs a different prime number, won't we run out?
  - About 10^{151} prime numbers < 512 bits
- What is two people accidentally pick the same prime number?
- If someone creates a database of all primes, won't he be able to use the database to break public-key algorithms?

Implementing RSA, cont.

- Break input into numerical blocks smaller than n.
- Encryption and decryption
  - Modular exponentiation

Security of RSA

- Rests on difficulty of factoring large numbers.
- If factoring large integers is easy, breaking RSA is easy
- Converse unproven: it is **conjectured** that if factoring large numbers is hard, breaking RSA is hard.
- Sure as hell better make sure n is a very very big number.

Factoring

Factoring a number means finding its prime factors.

10 = 2 * 5

60 = 2 * 2 * 3 * 5

833816926455584605284210207 1 =

179424673 * 2038074743 * 22801763489

To factor a number n, best known algorithm (number field sieve) has exponential running time

$$e^{c (\ln n)^{1/3} (\ln \ln n)^{2/3}}$$

A bit of factoring history…

- 1971 The big news was the factoring of a 41 digit number.
- 1991 RSA Data Security Inc set up RSA Factoring Challenge: list of hard numbers, each product of 2 primes, ranging from 100 digits to 500 digits.
- 1994 "RSA129", one of challenge numbers, 129 digits (428) bits), factored over 8 months, using 1600 computers on Internet around the world (~5000 MIPS-years)

  "We conclude that commonly used 512-bit RSA moduli are vulnerable to any organization prepared to spend a few million dollars and to wait a few months."
- With this method, a 250-digit number would take 100,000,000 times as long.

Chosen Ciphertext Attack Against RSA: Scenario 1

- Eve collects c. She needs m, for which $m = c^d$
- she chooses random r < n, she gets Bob's public key and then computes

  $x = r^e \bmod n \Rightarrow x^d = r^{ed} \bmod n$

  $\Rightarrow x^d = r \bmod n$

  $y = xc \bmod n$

  $t = r^{-1} \bmod n$

Eve gets Bob to decrypt y with his private key. Bob sends Eve

$u = y^d \bmod n$.

Now Eve computes

$tu \bmod n = r^{-1} y^d \bmod n = r^{-1} x^d c^d \bmod n$

$= c^d \bmod n = m$.

Chosen Ciphertext Attack Against RSA: Scenario 2

- Trent is a computer notary public. When Alice wants a document notarized, she sends it to Trent who signs it with an RSA digital signature.
- Mallory wants Trent to sign a message he otherwise wouldn't, call it m'
- Mallory chooses arbitrary x and computes $y = x^e \bmod n$ (where e is Trent's public key).
- Then he computes $m = ym' \bmod n$ and sends m to Trent to sign.
- Trent returns $m^d \bmod n = (ym')^d \bmod n = xm'^d \bmod n$.
- Mallory calculates $(m^d \bmod n) x^{-1} \bmod n = m'^d \bmod n$, which is the signature of m'.

What's going on?

- $(xm)^d \bmod n = x^d m^d \bmod n$

General Comments About Public-Key Cryptosystems

- Slow.
- Vulnerable to chosen-plaintext attacks.

Hybrid Cryptosystems

- In practice, public-key crypto used to secure and distribute **session keys**, which are then used with private-key crypto to secure message traffic.

- Alice gets ahold of Bob's public key.
- Alice generates random session key K, encrypts it using Bob's public key, and sends it to Bob.
- Bob decrypts Alice's message using his private key to recover session key.
- Both encrypt their communications using same session key.

Public-key crypto solves important key-management problem.

Digital Signatures

Digital Signatures

- Hand-written signatures used as proof of authorship or agreement with contents of a document.
  - authentic
  - unforgeable
  - not reusable
  - unalterable
  - cannot be repudiated.

- Not so obvious how to do on a computer.
  - Trivial to copy, cut and paste, easy to modify,….

Digital Signatures

- Two components:
  - secret signing algorithm S_k (M) = Signature
  - public verification algorithm V_k (M, Signature)
    V_k (M, Signature)= true if S_k(M) = Signature,
    false otherwise.

Signing Documents with Public Key Cryptography

- How Alice sends a **signed** message to Bob using RSA
  - Alice signs with her private key. S (M) = D_A(M)
  - Bob verifies with Alice's public key. V (C) = E_A(C)

A B

C = D_A(M) C

----------> M = E_A(C)

- Authentic, unforgeable, not reusable, unalterable, can't be repudiated.

Digital Signatures

- => can use RSA public-key cryptosystem to provide digital signatures.
- There are many other digital signature schemes:
  - Discrete Log Signature Schemes.
  - DSA
  - .....

Problem

- Copy of signed digital message identical to original.
  - Bob can cheat by reusing document and signature together.
  - Example: Alice sends Bob a signed digital check for $1000.
- Solution: timestamp.

Just to be completely clear… Using RSA…

- How Alice sends a **secret** message to Bob

  A B

  $C = E_B(M)$ C

  ----------> $M = D_B(C)$

- How Alice sends a **signed** message to Bob

  A B

  $C = D_A(M)$ C

  ----------> $M = E_A(C)$

Digital Signatures useful for

- Authentication: protocol by which the receiver of a message is convinced of the identity of the sender and the integrity of the message.

Digital Signatures + Encryption
proof of authorship + privacy

- Alice signs with her private key then encrypts with Bob's public key

A B

$C = E_B(S_A(M))$ C

----------->

- Bob decrypts with his private key, then verifies with Alice's public key.

$S_A(M) = D_B(C)$

$M = V_A(C)$

Issues

- Bad idea to encrypt then sign.
- Might be good idea to have different public-key/private-key pairs for signature and encryption.
- Timestamps should be used to prevent reuse of messages.

One Way Hash Functions

Problem with Signature Schemes

- Only "small" messages can be signed.
- A possible solution:
  - break long message into chunks and sign each chunk independently.
- Problems with this:
  - "secure" signature schemes are slow.
  - Various chunks of signed message could be rearranged/removed and resulting message would still be verified => want signature scheme to fail unless it is a signature of entire document.

Another Problem

- Perhaps the document that you're signing was sent to you by some bad guy.
  => remember chosen ciphertext attack against RSA!

A solution: One-way hash functions.

message ----------> message digest, fingerprint

M ----------> h(M)

arbitrarily long at most m bits

such that:

- Given M, easy to compute h(M)
- Given F, hard to compute M, such that h(M) = F
- Given M, hard to find another message M', s.t. h(M) = h(M')
- Hard to find two random messages M and M' such that h(M) = h(M')

Signatures and One-way Hash Functions

- How Alice sends a **signed** message to Bob

  A B

  $F = h(M)$

  $C = S_A(F)$ C, M

  ----------> if $V_A(C) = h(M)$

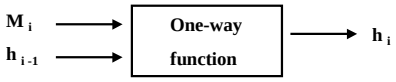
  signature verified.

Issues and Attacks

- If Mallory can find M′ s.t. h(M′)= h(M), could forge a message from Alice by sending C,M′
- If Mallory can find two message M′ and M″ s.t. h(M′)= h(M″) and convince Alice to sign h(M″), could forge C,M′.
- Birthday attack imposes a lower bound on the sizes of message digests.
  - Example: 40-bit message digest would be very insecure since a collision could be found with probability 1/2 with just over 2^{20} random hashes.
  - Secure Hash Standard: use 160 bit message digests.

Some One-Way Hash Functions

- MD4, MD5, SHS,…
- built using "compression function"



Key Exchange

Classic Secret Key Exchange Diffie-Hellman

- A and B agree on large prime p and g < p primitive mod p.
- r1 (selected by A) and r2 (B) are random large numbers

A		B
$X = g^{r1} \bmod p$	------- X -------->	
	<------Y--------	$Y = g^{r2} \bmod p$
$K = Y^{r1} \bmod p$		$K = X^{r2} \bmod p$

- K is secret key that both A and B have computed independently.

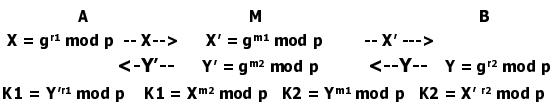
Security Against Eve

- DH Problem: Given $g^{r1} \bmod p$, $g^{r2} \bmod p$, for r1 and r2 chosen at random in 1..p-1, compute

  $g^{r1\ r2} \bmod p$

- If discrete logarithm problem is easy, so is DH Problem.
- Believed to be computationally intractable

Security Against Mallory Man-in-the-Middle Attack



- If they are communicating over a network in presence of active attacker, man-in-the-middle will always be a problem unless A and B share some information unavailable to Mallory.
- "Information advantage" -- some info predistributed over a trusted channel.

What we want

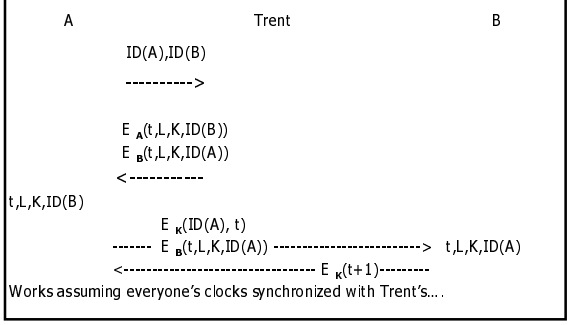
- Authenticated key exchange -- Alice and Bob exchange a secret key and at the same time each is sure they are talking to the other and not to Mallory.
- Many approaches, much confusion:
  - symmetric crypto (Kerberos)
  - public-key crypto

Authenticated Key Exchange Private-Key Setting

- Example: Kerberos
- Assumptions:
  - A and B each share secret keys with Trent (trusted authority/authentication server)
  - A wants to generate a session key for a conversation with Bob.

Kerberos

A key(T,A) **B** key(T,B)
t timestamp **K** session key **L** lifetime



Works assuming everyone's clocks synchronized with Trent's....

Authenticated Key Exchange--Public-Key Setting

- Secure publicly readable database protected from write-access by anyone but Trent.
- Trent is the Key Certification Authority or Key Distribution Center.
- Trent signs each public key with his private key. Everyone knows Trent's public key.
- In database, for each user, a compound message consisting of user's name, public key and other info signed by Trent (a certificate)

  $C(A) = (ID(A), V_A, S_T(ID(A), V_A))$

And then...

- Alice and Bob get each other's public keys from Trent (KDC)
- Alice generates random session key K

  A B

  $C = E_{KB} S_{KA}(K)$ C

  ----------> $K = D_{KB} V_{KA}(C)$

- Problems?

Message Authentication Codes

So far

- Cryptosystems ----> privacy, secrecy
- Digital Signatures ---> authentication
- Key Exchange with Authentication ----> handing out public keys and exchanging private keys in a trustworthy way.
- Message Authentication Codes (MAC) -----> protect data integrity when communicating in private key setting

MAC

- One-way hash function with secret key.

  $h_K(M) = F$
- Same properties as one-way hash functions.
- Additional property: only someone with key can verify hash.
- Useful for providing authenticity/data integrity without secrecy.

Some Applications of MACs

- A and B communicating using symmetric crypto: Use MAC to prevent Mallory from garbling encrypted data => send M, $h_K(M)$.

  Upon receipt of message, verify the MAC against message.

  Mallory can't garble messages without being caught because he can't compute the MAC.
- can be used by a single user to determine if her files have been altered, say by virus. User computes MAC of her files and stores value in table. If she used one-way hash function instead, virus could compute the new hash value after infection and replace the table entry.

Constructing a MAC

- One easy way: encrypt hash value with a symmetric algorithm.

Application: SSL

Secure Socket Layer (SSL) used by Navigator/Explorer…

- For example, when I send my credit card over the Internet to amazon.com

SSL (Step 1)

- Client (my Netscape browser)
  - says hello
  - sends random string R
  - describes private-key crypto supported (e.g. 40 bit RC4)
  - describes key-exchange protocol supported (e.g., RSA)

SSL Step 2

- Server (amazon.com) responds with
  - its own random string R'
  - its public key certified by Trent (e.g., Verisign)
  - requests that client authenticate by providing certificate from Trent (optional)

SSL Step 3

- Client responds by
  - generating a random secret key, called pre-Master Secret.
  - encrypts Pre-Master Secret using servers public key and sends it
  - sends certificate for her own public key
  - signs fingerprint of everything exchanged so far (including random strings R and R') and sends it.

SSL Step 4

- Both sides derive Master Secret from Pre-Master Secret, R and R'.
- Server then
  - derives MAC from Master Secret
  - sends client MAC of everything that came before.

SSL Step 5

- Both sides take Pre-Master Secret and derive encryption key for each direction and a MAC key for each direction.
- Every subsequent message is encrypted and MACed with sequence number.
- Typical encryption scheme: stream cipher like RC4 -- XOR plaintext with pseudo-random string. Very fast.

Conclusions on Crypto

- Tricky but fascinating business
- Not a field for amateurs