Lecture 6
Public Key Encryption

Review

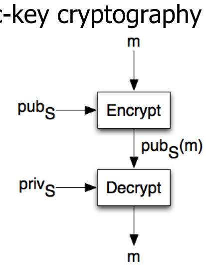
- The *key* in traditional crypto is used to encode the substitution rule
  - Needed to encrypt and decrypt
  - DES and Triple-DES use this technique
- Key distribution is the weak link
  - Hard to revoke
  - Disastrous if "codebook" is compromised
  - Hard to distribute (requires initial out-of-band secure exchange)
  - Impossible for the Web
- All of this changed in the 1970s…

Public-Key Cryptography

- Secure comm *without* key exchange
- Each party has a pair of related keys: public and private
  - A public key is published freely
  - A private key is shared with no one
- A message encrypted with one can be decrypted with the other
- You can't derive one from the other
  - This is critical!
- Original idea due to Diffie, Hellman, but RSA (Rivest, Shamir, Adelman) popular

Two Modes

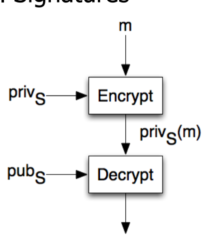
- Public-key cryptography



- Anyone can encrypt; only S can decrypt
- Used for data confidentiality

Two Modes

- Digital Signatures



- Only S can encrypt; anyone decrypts
- Used for authenticity

How Does it Work?

- Public key cryptography relies on so-called *trapdoor functions*
  - A fn that is easy to compute, but hard to invert without special info
  - "Easy" and "hard" meant computationally
- Some poor choices for trapdoor fns:
  - Add 2; Multiply by 3
- In practice quite difficult to find good trapdoor functions
- Most popular one is related to *prime factorization;* others possible

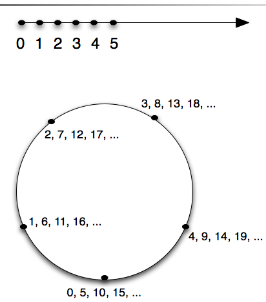
A Trapdoor Framework

- We want a fn *trapdoor*, s.t.:
  - $trapdoor_{pub}(s)$ fast to compute!
  - $trapdoor^{-1}_{pub}(s)$ *very, very slow* to compute
    - Heat death of universe slow
  - $trapdoor^{-1}_{pub}(s, priv)$ fast to compute
- For current choices of trapdoor, no proof that $trapdoor^{-1}_{pub}(s)$ is slow
  - We simply lack efficient algorithms today
  - Major algo breakthrough could ruin crypto

Prime Factorization

- n = p*q, where p and q are primes
  - Given p and q, easy to compute n
  - Given n, very hard to find p and q
- How can we turn this into a crypto system?
- We need two theorems. First…

INTERMISSION



- a = b + km (mod m)

Fermat's Little Theorem

- For any prime p, and any natural number a: $\mathbf{a^p = a \pmod p}$

a	p	A^p	A^p mod p	A mod p
4	3	64	1	1

Fermat's Little Theorem

- For any prime p, and any natural number a: $\mathbf{a^p = a \pmod p}$

a	p	A^p	A^p mod p	A mod p
4	3	64	1	1
4	5	1024	4	4

Fermat's Little Theorem

- For any prime p, and any natural number a: $\mathbf{a^p = a \pmod p}$

a	p	A^p	A^p mod p	A mod p
4	3	64	1	1
4	5	1024	4	4
7	11	1977326743	7	7

Fermat's Little Theorem

- For any prime p, and any natural number a: $a^p = a \pmod p$

a	p	A^p	A^p mod p	A mod p
4	3	64	1	1
4	5	1024	4	4
7	11	1977326743	7	7
7	12	13841287201	1	7

13

Fermat's Little Theorem

- Put another way, if a is not a multiple of p, then $a^{p-1} = 1 \pmod p$

14

Chinese Remainder Theorem

- Consider $x = a_i \pmod{p_i}$ for i=1,…k
- CRT says there's a solution for x if each of the p_i is relatively prime to every other p_j
- Also, solution is $x = 1 \pmod{\prod p_i}$

X = 1 (mod 4)
X = 1 (mod 5)
X = 1 (mod 20)

15

OK, back to crypto

- We choose two large primes, *p, q*
- n = pq
- Next:
  - Set $\lambda = (p-1)(q-1)$
  - Choose e, s.t. $e < \lambda$
  - Choose d, s.t. $de = 1 \pmod{\lambda}$
- n and e serve as public key
  - n is product of primes p, q
  - choosing e requires knowing p, q
- n and d serve as private key
  - Choosing d requires e and λ

16

OK, back to crypto

- $encrypt_{n,e}(m) = m^e \pmod n = c$
- $decrypt_{n,d}(c) = c^d \pmod n = m$
- Will decryption always work?
  - $c^d = m^{de}$
  - $m^{de} = m^{k\lambda+1}$
  - $m^{k\lambda+1} = m * (m^{(p-1)(q-1)})^k$
  - Recall from Fermat that
    - $m^{p-1} = 1 \pmod p$
    - $m^{q-1} = 1 \pmod q$
  - Use the CRT
    - $m^{(p-1)(q-1)} = 1 \pmod n$, where n = pq

17

OK, back to crypto

- $m^{k\lambda+1} = m * (m^{(p-1)(q-1)})^k$
- Recall from Fermat that
  - $m^{p-1} = 1 \pmod p$
  - $m^{q-1} = 1 \pmod q$
- Use the CRT
  - $m^{(p-1)(q-1)} = 1 \pmod n$, where n = pq
- $C^d = m * (m^{(p-1)(q-1)})^k$
- $C^d = m * (1)^k \pmod n$
- $C^d = m \pmod n$
- Thus, ciphertext c is equal to msg m

18

Review

- Encrypting and decrypting involve large exponentiation; not cheap, but doable
  - $encrypt_{n,e}(m) = m^e \pmod{n} = c$
  - $decrypt_{n,d}(c) = c^d \pmod{n} = m$
- Public, private keys require original primes to compute
  - Only product of primes is ever exposed
  - Computationally extremely challenging to recover original primes

19

Data Integrity

- How to ensure message has not been intercepted, changed, resent?
- Digital signatures, but asymmetric encryption CPU-expensive
- Solution:
  - Use a one-way function, a *hash function*
  - Any change to message will change hash
  - Sign the hash (which is small)
  - Append the hash to the message; if the receiver cannot reproduce the hash value, then attacker tampered with msg

20

MD5 Hash Algorithm

- Divide message into 512-bit blocks
- Create a digest (hash) for each block, plus final 128-bit "digest of digests"

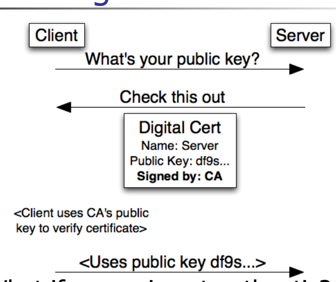
21

Public Key Infrastructure

- How do you get a public key?
  - Read it out of the phone book, off a billboard, off a business card, from an email signature line
  - But there are *lots* of possible public keys
- What if the public key is faked?
  - Attacker distributes a fake public key for B
  - A sends msg to B, encrypted with fake key
  - Attacker uses own private key to decrypt
- The PKI distributes public keys safely

22

PKI Design



- What if server is not authentic?
- How can we verify the certificate?
- Where is the weakest link?

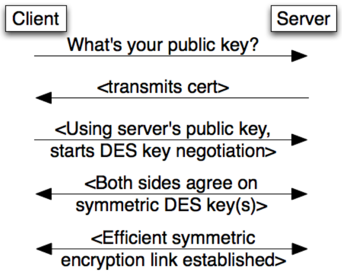
23

Certificate Authorities

- Verify identities and public keys
- Public keys for big CAs (Verisign, Thawte, lots of others) are built into browsers
- There can be a chain of cert signing
- You can start signing certs today! But you probably won't be built into Firefox
- Different cert "strengths" depending on level of identity verification

24

State of Play



- Note: CA never directly involved; little public-key-encrypted data

25

TLS/SSL

- Transport layer security / secure-sockets layer
- Commonly, **`https://`**
- Encryption of all content that goes into TCP payload

26