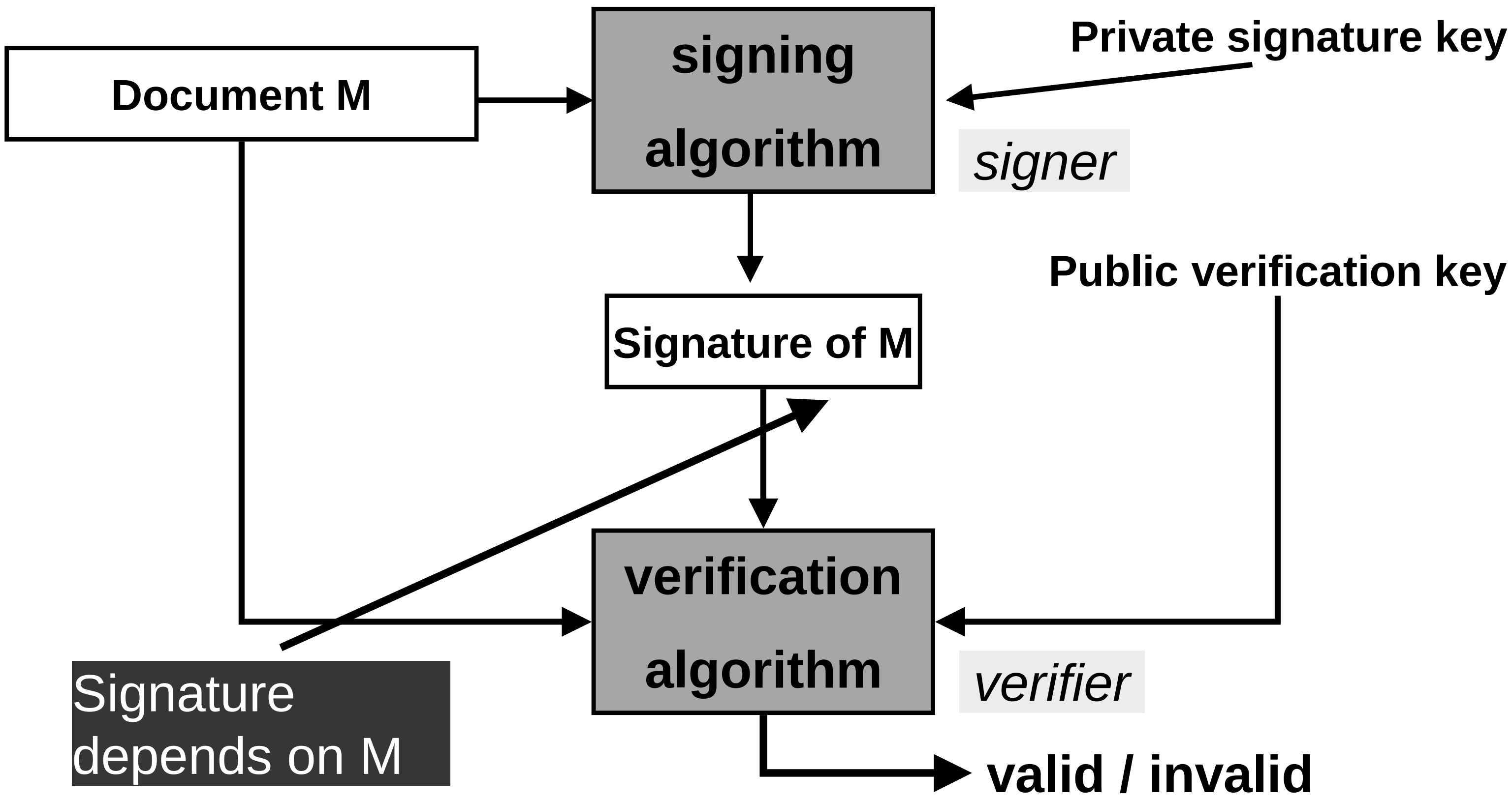
Introduction to Cryptography
Lecture 12

El Gamal signature,
Public Key Infrastructure (PKI),
some issues in number theory

Benny Pinkas

Signing/verification process



RSA with a full domain hash function

- Signature is sig(m) = $f^{-1}(H(m)) = (H(m))^d$ mod N.
  - $H()$ is such that its range is $[1,N]$
- *The system is no longer homomorphic*
  - $sig(m) \cdot sig(m') \neq sig(m \cdot m')$
- *Seems hard to generate a random signature*
  - Computing s^e is insufficient, since it is also required to show m s.t. $H(m) = s^e$.
- Proof of security in the random oracle model – where $H()$ is modeled as a random function

El Gamal signature scheme

- Invented by same person but different than the encryption scheme. (think why)
- A randomized signature: same message can have different signatures.
- Based on the hardness of extracting discrete logs
- The DSA (Digital Signature Algorithm/Standard) that was adopted by NIST in 1994 is a variation of El-Gamal signatures.

El Gamal signatures

- Key generation:
  - Work in a group Z_p^* where discrete log is hard.
  - Let g be a generator of Z_p^*.
  - Private key $1 < a < p-1$.
  - Public key $p, g, y=g^a$.

- Signature: (of M)
  - Pick random $1 < k < p-1$, s.t. $\gcd(k,p-1)=1$.
  - Compute $m=H(M)$.
    - $r = g^k \bmod p$.
    - $s = (m - r \cdot a) \cdot k^{-1} \bmod (p-1)$
  - Signature is r, s.

El Gamal signatures

- Signature:
  - Pick random $1 < k < p-1$, s.t. gcd$(k,p-1)=1$.
  - Compute
    - $r = g^k$ mod p.
    - $s = (m - r \cdot a) \cdot k^{-1}$ mod $(p-1)$
- Verification:
  - Accept if
    - 0 < r < p
    - $y^r \cdot r^s = g^m$ mod p

same *r* in
both places!

- It works since $y^r \cdot r^s = (g^a)^r \cdot (g^k)^s = g^{ar} \cdot g^{m-ra} = g^m$
- Overhead:
  - Signature: one (offline) exp.   Verification: three exps.

El Gamal signature: comments

- Can work in any finite Abelian group
  - The discrete log problem appears to be harder in elliptic curves over finite fields than in Z_p* of the same size.
  - Therefore can use smaller groups $\Rightarrow$ shorter signatures.
- Forging: find $y^r \cdot r^s = g^m \bmod p$
  - E.g., choose random $r = g^k$ and either solve dlog of g^m/y^r to the base r, or find $s=k^{-1}(m - \log_g y \cdot r)$ *(????)*
- Notes:
  - A different k must be used for every signature
  - If no hash function is used (i.e. sign M rather than $m=H(M)$), existential forgery is possible
  - If receiver doesn't check that $0<r<p$, adversary can sign messages of his choice.

Trusting public keys

- Public key technology requires every user to remember its private key, and to have access to other users' public keys
- How can the user verify that a public key PK_v corresponds to user *v*?
  - What can go wrong otherwise?
- A simple solution:
  - A trusted public repository of public keys and corresponding identities
    - Doesn't scale up
    - Requires online access per usage of a new public key

Certification Authorities (CA)

- A method to bootstrap trust
  - Start by trusting a single party and knowing its public key
  - Use this to establish trust with other parties (and associate them with public keys)

- The Certificate Authority (CA) is trusted party.
  - All users have a copy of the public key of the CA
  - The CA signs Alice's digital certificate. A simplified certificate is of the form *(Alice, Alice's public key)*.

Certification Authorities (CA)

- When we get Alice's certificate, we
  - Examine the identity in the certificate
  - Verify the signature
  - Use the public key given in the certificate to
    - Encrypt messages to Alice
    - Or, verify signatures of Alice
- The certificate can be sent by Alice without any online interaction with the CA.

Certificates

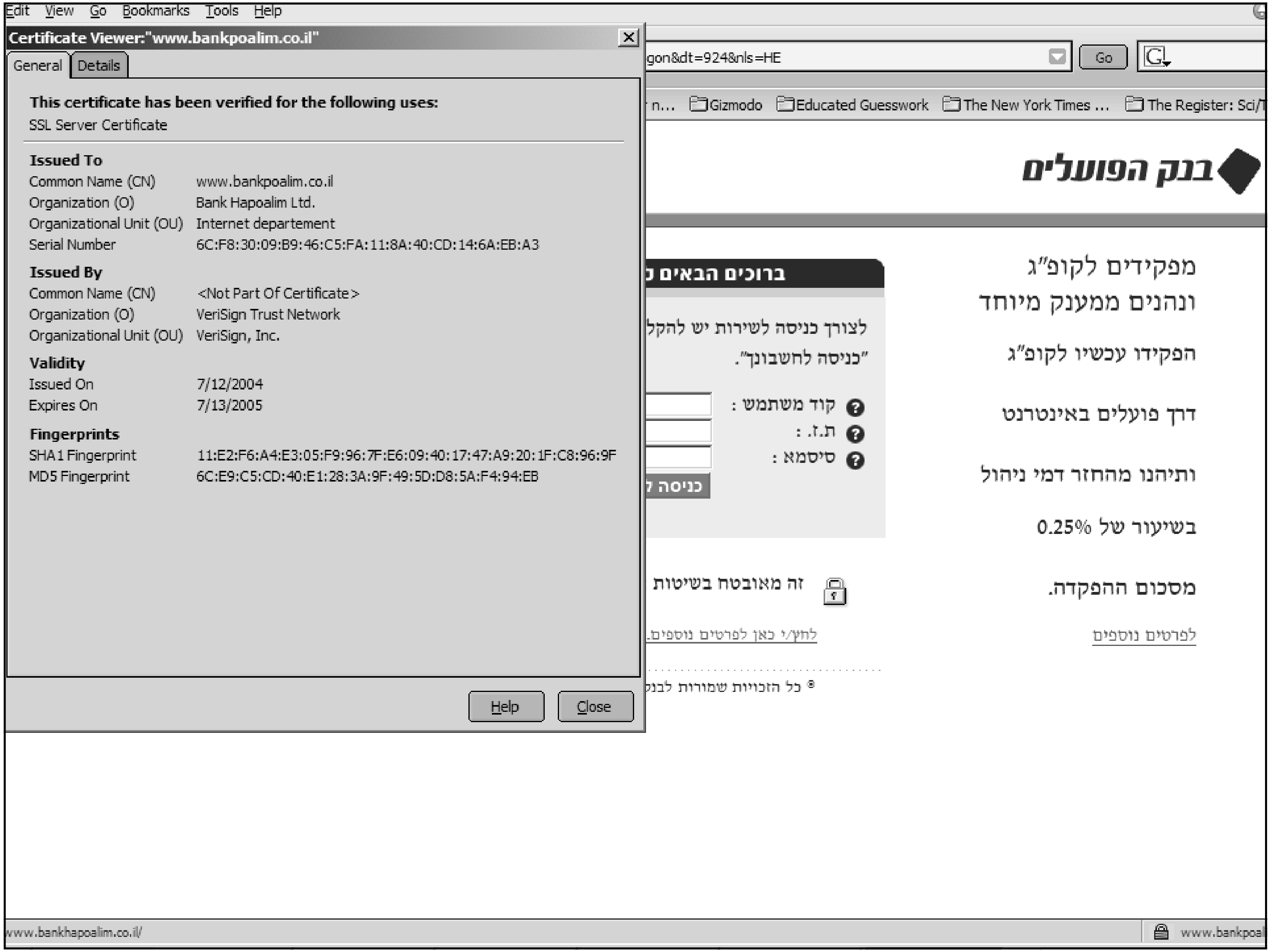
- A certificate usually contains the following information
  - Owner’s name
  - Owner’s public key
  - Encryption/signature algorithm
  - Name of the CA
  - Serial number of the certificate
  - Expiry date of the certificate
  - …
- Your web browser contains the public keys of some CAs
- A web site identifies itself by presenting a certificate which is signed by a chain starting at one of these CAs

Certification Authorities (CA)

- Unlike KDCs, the CA does not have to be online to provide keys to users
  - It can therefore be better secured than a KDC
  - The CA does not have to be available all the time
- Users only keep a single public key – of the CA
- The certificates are not secret. They can be stored in a public place.
- When a user wants to communicate with Alice, it can get her certificate from either her, the CA, or a public repository.
- A compromised CA
  - can mount active attacks (certifying keys as being Alice's)
  - but it cannot decrypt conversations.

An example of an X.509 certificate

```
Certificate:
  Data:
    Version: 1 (0x0)
    Serial Number: 7829 (0x1e95)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
      OU=Certification Services Division, CN=Thawte Server
      CA/emailAddress=server-certs@thawte.com
    Validity
         Not Before: Jul 9 16:04:02 1998 GMT
         Not After : Jul 9 16:04:02 1999 GMT
    Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala, OU=FreeSoft,
      CN=www.freesoft.org/emailAddress=baccala@freesoft.org
    Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
         RSA Public Key: (1024 bit)
         Modulus (1024 bit): 00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
           33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
           66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
           70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
           16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
           c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
           8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
           d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8: e8:35:1c:9e:27:52:7e:41:8f
         Exponent: 65537 (0x10001)
  Signature Algorithm: md5WithRSAEncryption
    93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
      92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:…
```


Edit View Go Bookmarks Tools Help
Certificate Viewer:"www.bankpoalim.co.il"
General Details
This certificate has been verified for the following uses:
SSL Server Certificate
Issued To
Common Name (CN) www.bankpoalim.co.il
Organization (O) Bank Hapoalim Ltd.
Organizational Unit (OU) Internet departement
Serial Number 6C:F8:30:09:B9:46:C5:FA:11:8A:40:CD:14:6A:EB:A3
Issued By
Common Name (CN) <Not Part Of Certificate>
Organization (O) VeriSign Trust Network
Organizational Unit (OU) VeriSign, Inc.
Validity
Issued On 7/12/2004
Expires On 7/13/2005
Fingerprints
SHA1 Fingerprint 11:E2:F6:A4:E3:05:F9:96:7F:E6:09:40:17:47:A9:20:1F:C8:96:9F
MD5 Fingerprint 6C:E9:C5:CD:40:E1:28:3A:9F:49:5D:D8:5A:F4:94:EB
Help Close
gon&dt=924&nls=HE
Go
Gizmodo
Educated Guesswork
The New York Times ...
The Register: Sci/T
בנק הפועלים
מפקידים לקופ"ג
ונהנים ממענק מיוחד
הפקידו עכשיו לקופ"ג
דרך פועלים באינטרנט
ותיהנו מהחזר דמי ניהול
בשיעור של 0.25%
מסכום ההפקדה.
לפרטים נוספים
קוד משתמש :
ת.ז. :
סיסמא :
www.bankhapoalim.co.il/

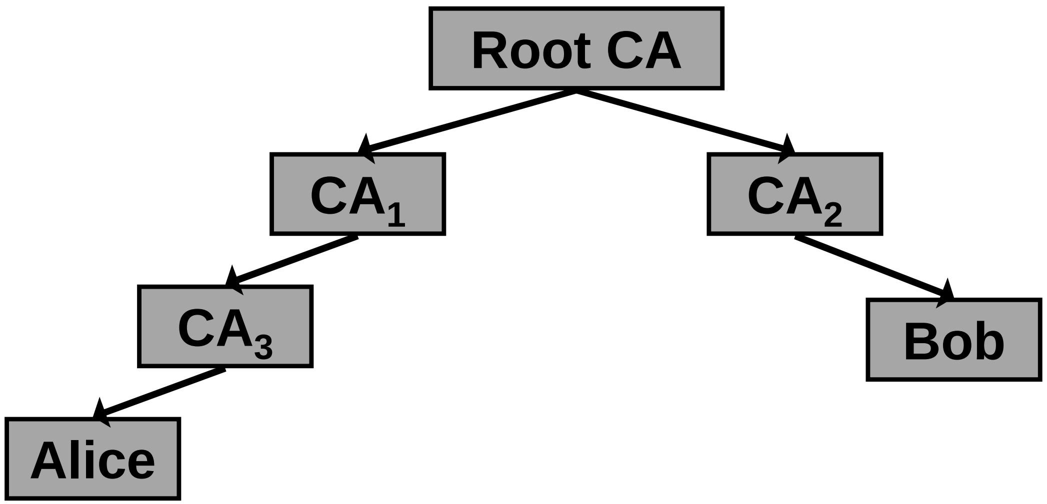
Public Key Infrastructure (PKI)

- The goal: build trust on a global level

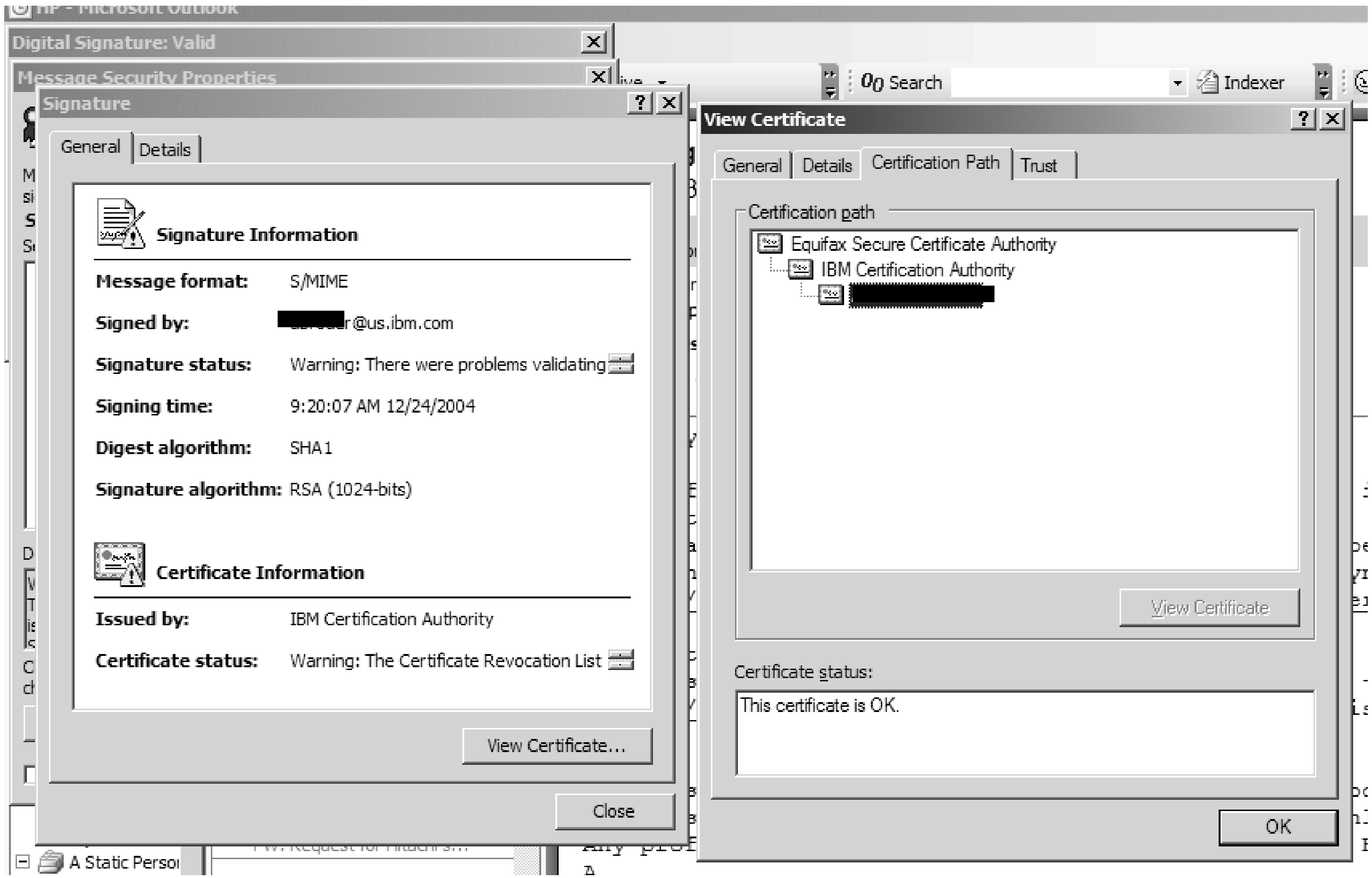
- Running a CA:
  - If people trust you to vouch for other parties, everyone needs you.
  - A license to print money
  - But,
    - The CA should limit its responsibilities, buy insurance…
    - It should maintain a high level of security
    - Bootstrapping: how would everyone get the CA's public key?

Public Key Infrastructure (PKI)

- Monopoly: a single CA vouches for all public keys
  - Suitable in particular for enterprises.
- Monopoly + delegated CAs:
  - top level CA can issue *speial* certificates for other CAs
  - Certificates of the form
    - [(Alice, $PK_A)_{CA3}$, (CA3, $PK_{CA3})_{CA1}$, (CA1, $PK_{CA1})_{ROOT\text{-}CA}$]



Certificate chain



Revocation

- Revocation is a key component of PKI
  - Each certificate has an expiry date
  - But certificates might get stolen, employees might leave companies, etc.
  - Certificates might therefore need to be revoked before their expiry date
  - New problem: before using a certificate we must verify that it has not been revoked
    - Often the most costly aspect of running a large scale public key infrastructure (PKI)
    - How can this be done efficiently?
    - (we won't discuss this issue this year)

SSL / TLS

SSL/TLS

- General structure of secure HTTP connections
  - To connect to a secure web site using SSL or TLS, we send an https:// command
  - The web site sends back a public key(1), and a certificate.
  - Our browser
    - Checks that the certificate belongs to the url we're visiting
    - Checks the expiration date
    - Checks that the certificate is signed by a CA whose public key is known to the browser
    - Checks the signature
    - If everything is fine, it chooses a session key and sends it to the server encrypted with RSA using the server's public key

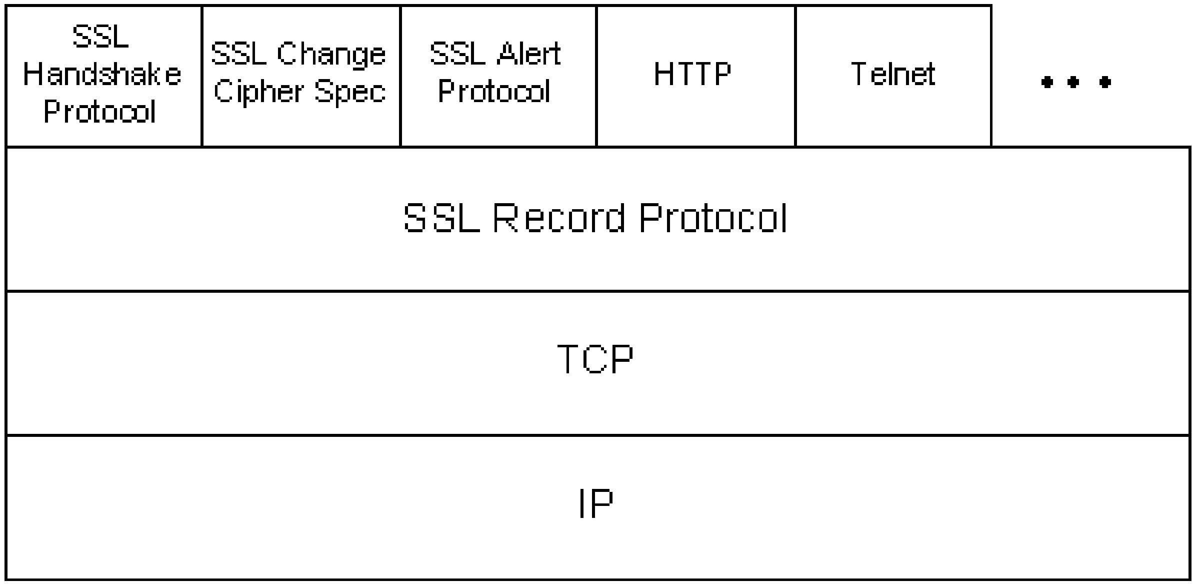
(1) This is a very simplified version of the actual protocol.

SSL/TLS

- SSL (Secure Sockets Layer)
  - SSL v2
    - Released in 1995 with Netscape 1.1
    - A flaw found in the key generation algorithm
  - SSL v3
    - Improved, released in 1996
    - Public design process
- TLS (Transport Layer Security)
  - IETF standard, RFC 2246
- Common browsers support all these protocols

SSL Protocol Stack

- SSL/TLS operates over TCP, which ensures reliable transport.
- Supports any application protocol (usually used with http).



SSL/TLS Overview

- Handshake Protocol - establishes a session
  - Agreement on algorithms and security parameters
  - Identity authentication
  - Agreement on a key
  - Report error conditions to each other

- Record Protocol - Secures the transferred data
  - Message encryption and authentication

- Alert Protocol – Error notification (including "fatal" errors).

- Change Cipher Protocol – Activates the pending crypto suite

Simplified SSL Handshake

Client

Server

I want to talk, ciphers I support, R_C

Certificate (PK_{Server}), cipher I choose, R_S

$\{S\}_{PKserver}$, {keyed hash of handshake message}

compute
$K = f\,(S,R_C,R_S)$

{keyed hash of handshake message}

compute
$K = f\,(S,R_C,R_S)$

Data protected by keys derived from K

A typical run of a TLS protocol

- $C \Rightarrow S$
  - ClientHello.protocol.version = “TLS version 1.0”
  - ClientHello.random = T_C, N_C
  - ClientHello.session_id = “NULL”
  - ClientHello.crypto_suite = “RSA: encryption.SHA-1:HMAC”
  - ClientHello.compression_method = “NULL”
- $S \Rightarrow C$
  - ServerHello.protocol.version = “TLS version 1.0”
  - ServerHello.random = T_S, N_S
  - ServerHello.session_id = “1234”
  - ServerHello.crypto_suite = “RSA: encryption.SHA-1:HMAC”
  - ServerHello.compression_method = “NULL”
  - ServerCertificate = pointer to server’s certificate
  - ServerHelloDone

Some additional issues

- More on S $\Rightarrow$ C
  - The ServerHello message can also contain Certificate Request Message
  - I.e., server may request client to send its certificate
  - Two fields: certificate type and acceptable CAs

- Negotiating crypto suites
  - The crypto suite defines the encryption and authentication algorithms and the key lengths to be used.
  - ~30 predefined standard crypto suites
  - Selection (SSL v3): Client proposes a set of suites. Server selects one.

Key generation

- Key computation:
  - The key is generated in two steps:
  - *pre-master secret S* is exchanged during handshake
  - *master secret K* is a 48 byte value calculated using pre-master secret and the random nonces

- Session vs. Connection: a *session* is relatively long lived. Multiple TCP *connections* can be supported under the same SSL/TSL connection.
- For each connection: 6 keys are generated from the master secret *K* and from the nonces. (For each direction: encryption key, authentication key, IV.)

TLS Record Protocol

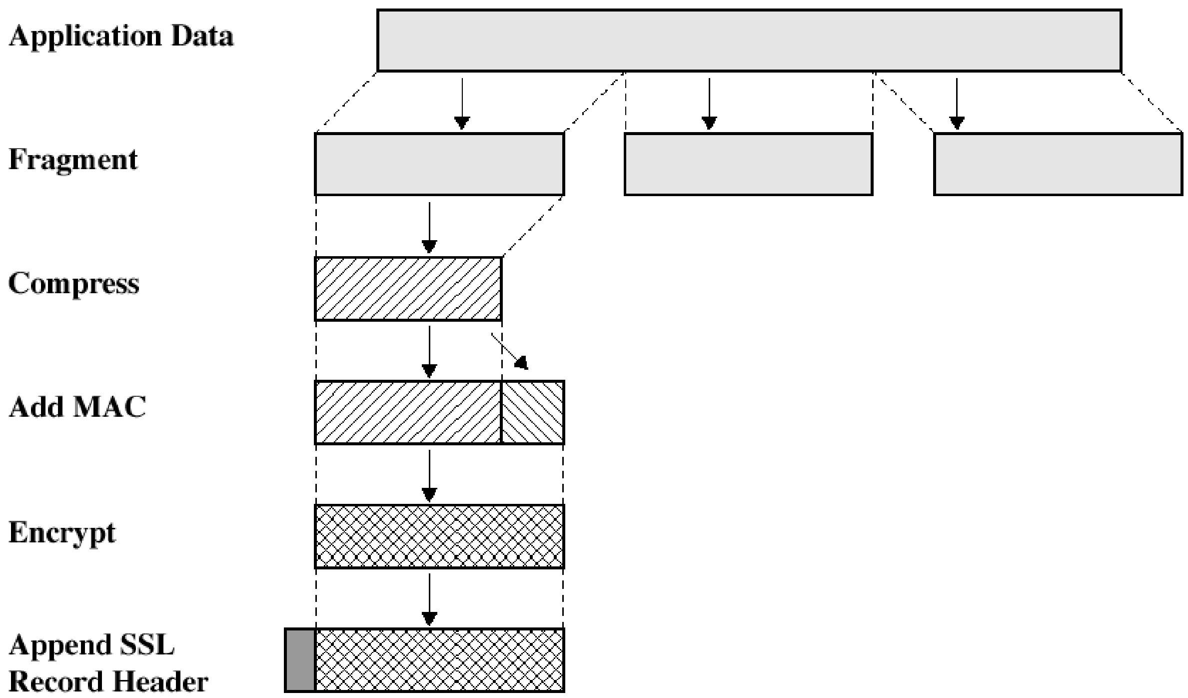


Figure 17.3 SSL Record Protocol Operation

Some practical issues in number theory

Primality testing

- Why do we need primality testing?
  – Essentially all public key cryptographic algorithms use large prime numbers
  – We therefore need an algorithm for prime number generation
  – Suppose we have an algorithm "PrimalityTest" with a binary output.
  – We can generate random primes as follows

```
GeneratePrime(a,b)
  1. Choose random number x ∈ [a,b]
  2. If PrimalityTest(x) then output "x is
  prime"; otherwise goto line 1.
```

Density of prime numbers

- How long will GeneratePrime run?
- Let $\pi(n)$ specify number of primes ≤ n.

- Prime number theorem:
  - $\pi(n)$ goes to $n / \ln n$ as n goes to infinity.
- Pretty accurate even for small n (e.g. for $n=2^{30}$ it is off by 6%).

- Corollary: a random number in [1,n] is prime with probability $1/\ln n$. (e.g. for $n=2^{512}$, probability is *1/355*).
  - The GeneratePrime algorithm is expected to take $\ln n$ rounds.
  - If we skip even numbers, we cut running time by ½.

Primality testing

- Primality testing is a decision problem: “is x prime or composite?”
- Different than the search problem “find all prime factors of x” (“factor x”).
- In this case, the decision problem has an efficient solution while the search problem does not.

- First algorithm: Trial division
  - Try to divide x by every prime integer smaller than $\sqrt{x}$ (sqrt(x)).
  - Infeasible for large x.

Fermat's test

- Fermat's theorem: if p is prime then for all $1 \leq a < p$ it holds that $a^{p-1} = 1$ mod p.

- If we can find an a s.t $a^{x-1} \neq 1$ mod x, then x is surely composite.
  - Surprisingly, the converse is almost always true, and for a large percentage of the choices of a.
  - Suppose we check only for $a=2$.
    - `If` 2^{x-1} `!= 1 mod x`

      ```
      –Then return COMPOSITE     /for sure
      –Otherwise, return PRIME   /we hope
      ```

  - How accurate is this program?

Fermat's test

- Surprisingly, this test is almost always right
  - Wrong for only 22 values of x smaller than 100,000
  - Probability of error goes down to 0 as x grows
    - For |x|=512 bits, probability of error is $< 10^{-20} \approx 2^{-66}$
    - For |x|=1024 bits, probability of error is $< 10^{-41} \approx 2^{-136}$
- The test is therefore sufficient for randomly chosen candidate primes
- But we need a better test if x is not chosen at random
- Cannot eliminate errors by checking for bases ≠ 2
  - x is a Charmichael number if it is composite, but $a^{x-1} = 1 \bmod x$ for all $1 \leq a < x$.
  - There are infinitely many Charmichael numbers
  - But they are very rare

Miller-Rabin test

- Works for all numbers (even Charmichael numbers).
  - Checks several randomly chosen bases a
  - If it finds out that $a^{x-1} = 1$ mod x, it checks whether the process found a nontrivial root of 1 $(\neq 1,-1)$. If so, it outputs COMPOSITE.

The Miller-Rabin test:

1. Write $x-1=2^c r$ for an odd r. set comp=0.
2. For i=1 to T
   - Pick random a $\in$ [1,x-1]. If gcd(a,x)> 1 set comp=1.
   - Compute $y_0=a^r$ mod x, $y_i=(y_{i-1})^2$ mod x for $i=1..c$. If $y_c \neq 1$, or $\exists i$, $y_i=1$, $y_{i-1} \neq \pm 1$, set comp=1.
3. If comp=1 return COMPOSITE, else PRIME.

Miller-Rabin test

- Possible values for the sequence $y_0=a^r, y_1=a^{2r} \ldots y_c=a^{x-1}$.
  - <…,d>, where $d \neq 1$, decide COMPOSITE.
  - <1,1,…,1>, decide PRIME.
  - <..,-1,1,..,1>, decide PRIME.
  - <…,d,1,…,1>, where $d \neq \pm 1$, decide COMPOSITE.
  - For a composite number x, we denote a base a as a non-witness if it results in the output being “PRIME”.
- Lemma: if x is an odd composite number then the number of non-witnesses is at most $x/4$.
- Therefore, for any odd integer x, T trials give the wrong answer with probability < $(1/4)^T$.

Breaking News

- Primes $\in$ P
  - Agrawal, Kayal, Saxena (2004)

Integer factorization

- The RSA and Rabin cryptosystems use a modulus N and are insecure if it is possible to factor N.

- Factorization: given N find all prime factors of N.

- Factoring is the search problem corresponding to the primality testing decision problem.
  - Primality testing is easy
  - What about factoring?

Modern factoring algorithms

- The number-theoretic running time function $L_n(a,c)$

$$L_n(a,c) = e^{c(\ln n)^a (\ln\ln n)^{1-a}}$$

  - For a=0, the running time is polynomial in ln(n).
  - For a=1, the running time is exponential in ln(n).
  - For 0<a<1, the running time is subexponential.

- Factoring algorithms
  - Quadratic field sieve: $L_n(1/2, 1)$
  - General number field sieve: $L_n(1/3, 1.9323)$
  - Elliptic curve method $L_p(1/2, 1.41)$ (preferable only if p<<sqrt(n))

Modulus size recommendations

- Factoring algorithms are run on massively distributed networks of computers (running in their idle time).
- RSA published a list of factoring challenges.
- A 512 bit challenge was factored in 1999.
- The largest factored number $n=pq$.
  - 640 bits (RSA-640)
  - Factored on November 2, 2005 using the NFS
- Typical current choices:
  - At least 1024-bit RSA moduli should be used
  - For better security, longer RSA moduli are used
  - For more sensitive applications, key lengths of 2048 bits (or higher) are used

Discrete log algorithms

- Input: *(g,y)* in a finite group G. Output: x s.t. $g^x = y$ in G.
- Generic vs. special purpose algorithms: generic algorithms do not exploit the representation of group elements.

- Algorithms
  - Baby-step giant-step: Generic. |G| can be unknown. Sqrt(|G|) running time and memory.
  - Pollard's rho method: Generic. |G| must be known. Sqrt(|G|) running time and O(1) memory.
  - No generic algorithm can do better than O(sqrt(q)), where q is the largest prime factor of |G|
  - Pohlig-Hellman: Generic. |G| and its factorization must be known. O(sqrt(q) ln q), where q is largest prime factor of |G|.
  - Therefore for Z^*_p, p-1 must have a large prime factor.
  - Index calculus algorithm for Z^*_p: L(1/2, c)
  - Number field size for Z^*_p: L(1/3, 1.923)

Elliptic Curves

- The best discrete log algorithm which works even if |G| can be unknown is the baby-step giant-step algorithm.
  - Sqrt(|G|) running time and memory.
- Other (more efficient) algorithms must know |G|.
  - In Z_p^* we know that | Z_p^* |=p-1.

- Elliptic curves are groups G where
  - The Diffie-Hellman assumption is assumed to hold, and therefore we can run DH an ElGamal encryption/sigs.
  - |G| is unknown and therefore the best discrete log algorithm us pretty slow
  - It is therefore believed that a small Elliptic Curve group is as secure as larger Z_p^* group.
  - Smaller group -> smaller keys and more efficient operations.

Baby-step giant-step DL algorithm

- Let t=sqrt(|G|).
- x can be represented as $x=ut-v$, where u,v < sqrt(|G|).

- The algorithm:
  – Giant step: compute the pairs $(j, g^{j \cdot t})$, for $0 \le j \le t$. Store in a table keyed by $g^{j \cdot t}$.
  – Baby step: compute $y \cdot g^i$ for $i=0,1,2\ldots$, until you hit an item $(j, g^{j \cdot t})$ in the table. x = $jt - i$.

- Memory and running time are O(sqrt|G|).

Baby-step giant-step DL algorithm