Introduction to Cryptography
Lecture 10

Signatures, Public Key Infrastructure (PKI), hash chains, hash trees, SSL.

Benny Pinkas

Some more signature schemes

Rabin signatures

- Same paradigm as RSA signatures:
  - $f(m) = m^2 \bmod N$. $(N=pq)$.
  - $Sig(m) = s$, s.t. $s^2 = m \bmod N$. I.e., the square root of m.

- *Unlike RSA,*
  - Not all m are QR mod N.
  - Therefore, only ¼ of messages can be signed.
- *Solutions:*
  - Use random padding. Choose padding until you get a QR.
  - Deterministic padding (Williams system).
- A *total break* given a chosen message attack. (show)
- Must therefore use a hash function H as in RSA.

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.

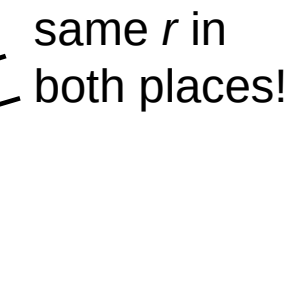
El Gamal signatures

- 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
    - $r = g^k \bmod p$.
    - $s = (m - r \cdot a) \cdot k^{-1} \bmod (p-1)$
- Verification:
  - Accept if
    - $0 < r < p$
    - $y^r \cdot r^s = g^m \bmod p$



- 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.

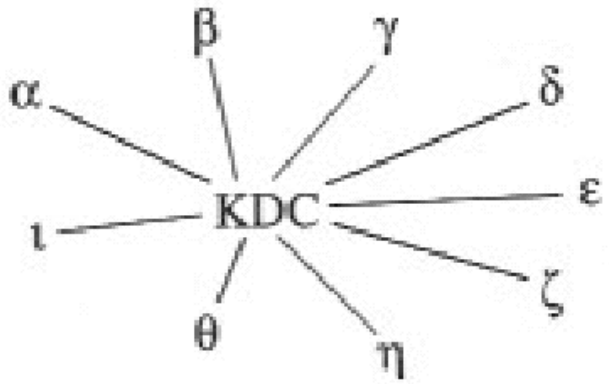
Public Key Infrastructure

Key Infrastructure for symmetric key encryption

- Each user has a shared key with each other user
  - A total of n(n-1)/2 keys
  - Each user stores n-1 keys

Key Distribution Center (KDC)

- The KDC shares a symmetric key K_u with every user *u*
- Using this key they can establish a trusted channel
- When *u* wants to communicate with *v*
  - *u* sends a request to the KDC
  - The KDC
    - *authenticates u*
    - *g*enerates a key K_{uv} to be used by *u* and *v*
    - *sends* $Enc(K_u, K_{uv})$ to *u*, and $Enc(K_v, K_{uv})$ to *v*



Key Distribution Center (KDC)

- Advantages:
  - A total of *n* keys, one key per user.
  - easier management of joining and leaving users.
- Disadvantages:
  - The KDC can impersonate anyone
  - The KDC is a single point of failure, for both
    - security
    - quality of service
- Multiple copies of the KDC
  - More security risks
  - But better availability

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.

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.

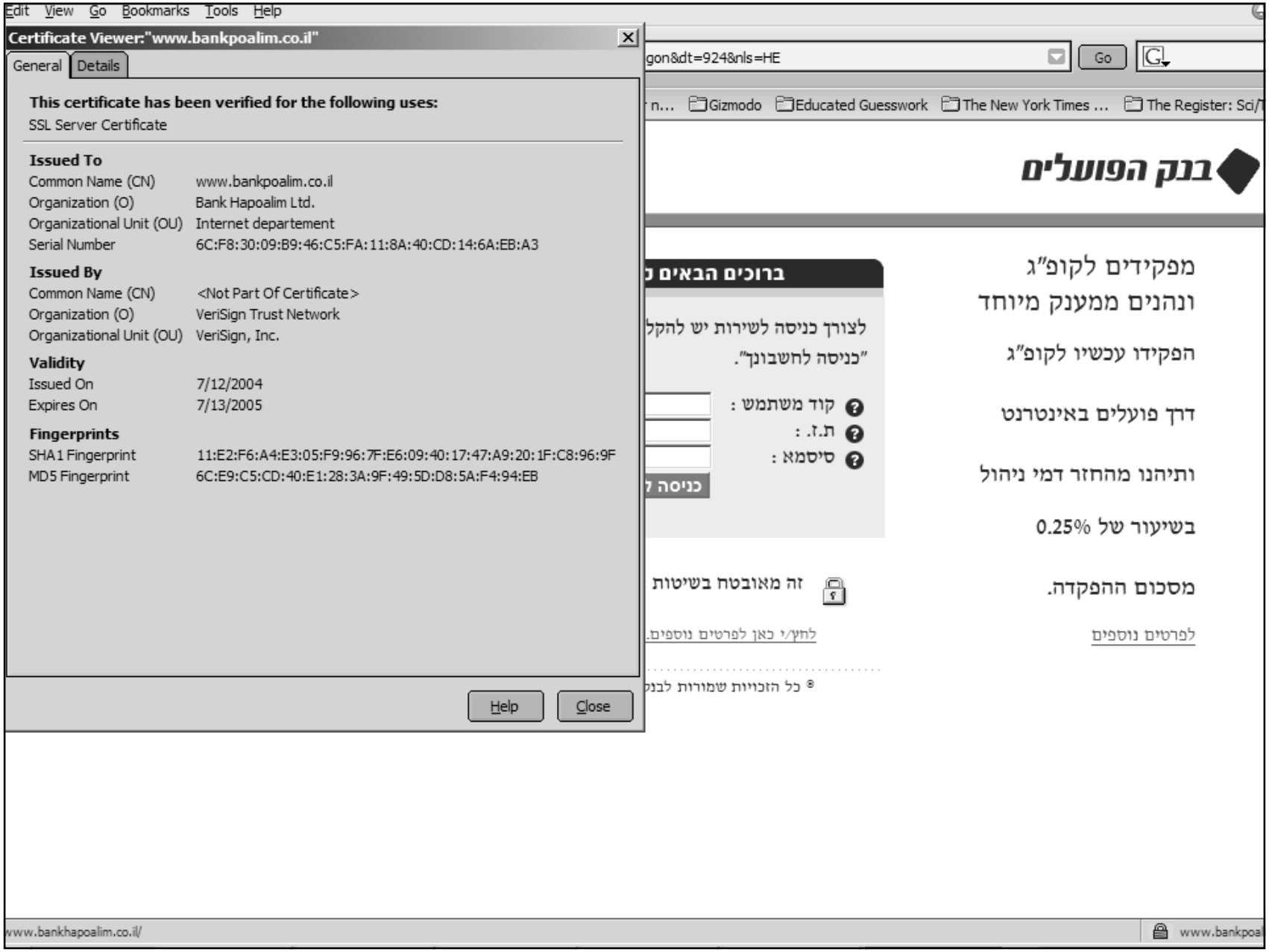
Certification Authorities (CA)

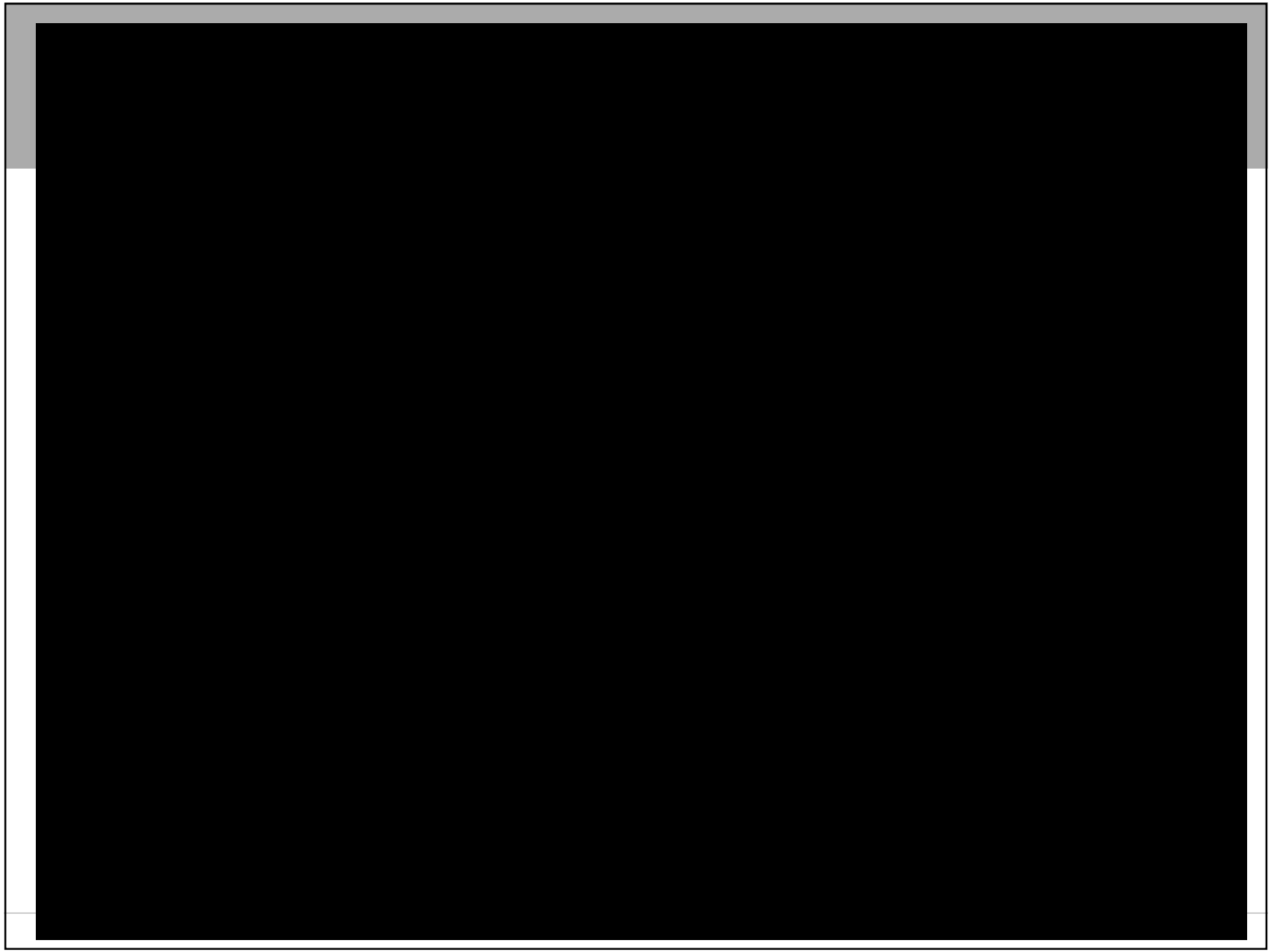
- An example.
  - To connect to a secure web site using SSL or TLS, we send an https:// command
  - The web site sends back a public key(1), and a certificate.
  - Our browser
    - Checks that the certificate belongs to the url we're visiting
    - Checks the expiration date
    - Checks that the certificate 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

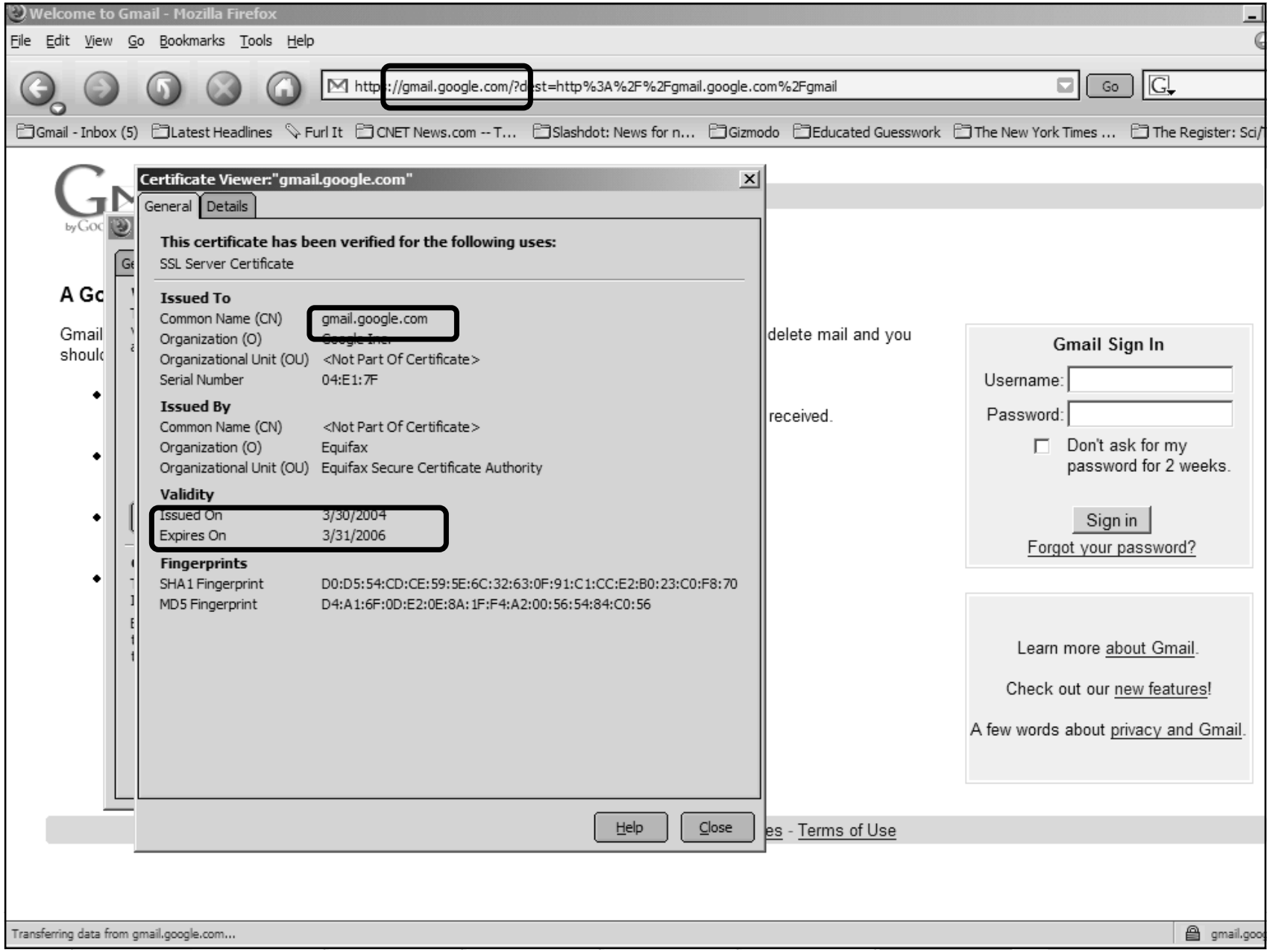
(1) This is a very simplified version of the actual protocol.

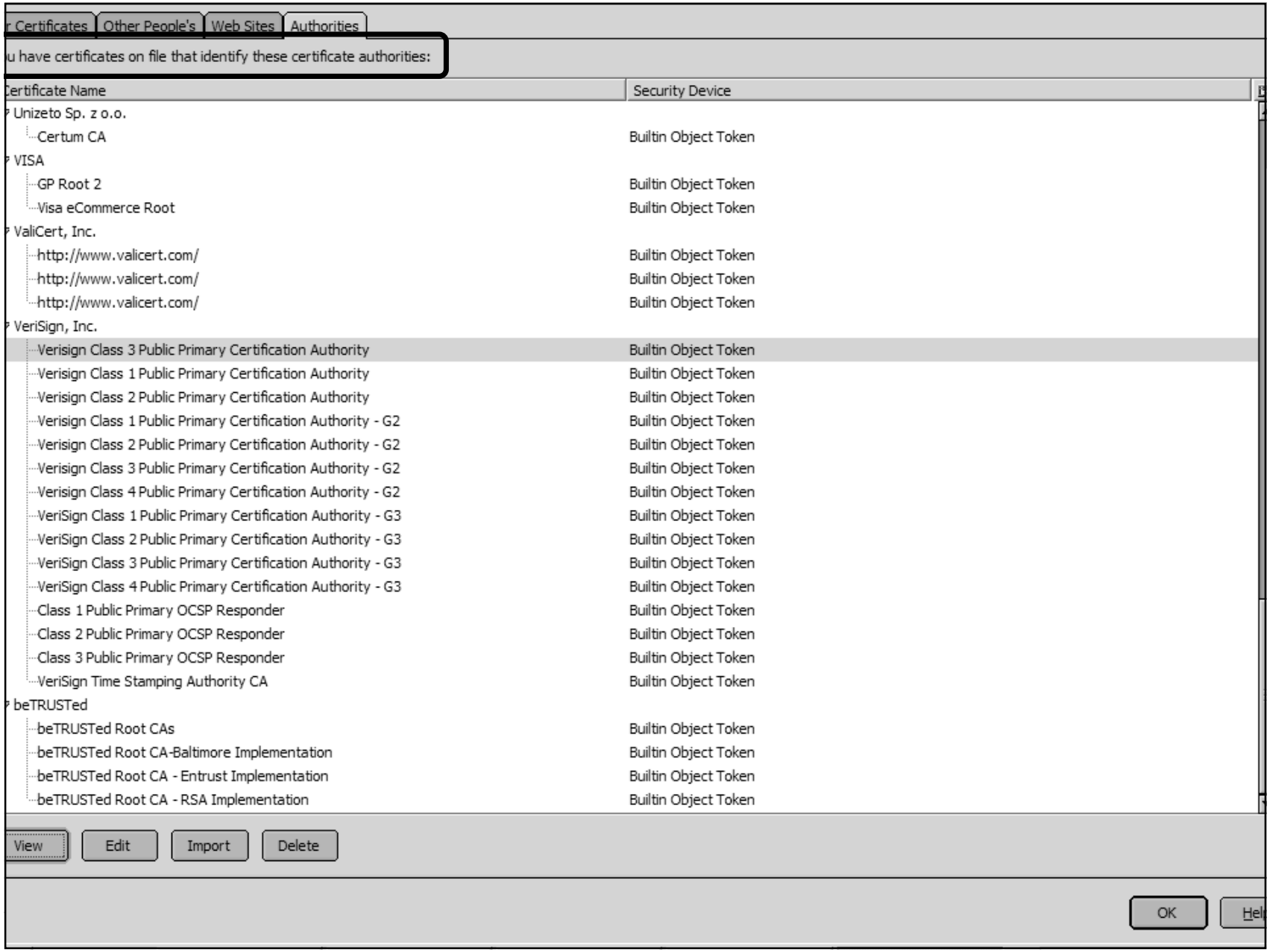
An example of an X.509 certificate

```
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
Go
Gizmodo
Educated Guesswork
The New York Times ...
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 :
זה מאובטח בשיטות
www.bankhapoalim.co.il/




Welcome to Gmail - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://gmail.google.com/?dest=http%3A%2F%2Fgmail.google.com%2Fgmail
Go
Gmail - Inbox (5)
Latest Headlines
Furl It
CNET News.com -- T...
Slashdot: News for n...
Gizmodo
Educated Guesswork
The New York Times ...
The Register: Sci/
Certificate Viewer:"gmail.google.com"
General
Details
This certificate has been verified for the following uses:
SSL Server Certificate
Issued To
Common Name (CN) gmail.google.com
Organization (O)
Organizational Unit (OU) <Not Part Of Certificate>
Serial Number 04:E1:7F
Issued By
Common Name (CN) <Not Part Of Certificate>
Organization (O) Equifax
Organizational Unit (OU) Equifax Secure Certificate Authority
Validity
Issued On 3/30/2004
Expires On 3/31/2006
Fingerprints
SHA1 Fingerprint D0:D5:54:CD:CE:59:5E:6C:32:63:0F:91:C1:CC:E2:B0:23:C0:F8:70
MD5 Fingerprint D4:A1:6F:0D:E2:0E:8A:1F:F4:A2:00:56:54:84:C0:56
Help
Close
delete mail and you
received.
Gmail Sign In
Username:
Password:
Don't ask for my password for 2 weeks.
Sign in
Forgot your password?
Learn more about Gmail.
Check out our new features!
A few words about privacy and Gmail.
es - Terms of Use
Transferring data from gmail.google.com...


r Certificates
Other People's
Web Sites
Authorities
u have certificates on file that identify these certificate authorities:
ertificate Name
Security Device
Unizeto Sp. z o.o.
Certum CA
Builtin Object Token
VISA
GP Root 2
Builtin Object Token
Visa eCommerce Root
Builtin Object Token
ValiCert, Inc.
http://www.valicert.com/
Builtin Object Token
http://www.valicert.com/
Builtin Object Token
http://www.valicert.com/
Builtin Object Token
VeriSign, Inc.
Verisign Class 3 Public Primary Certification Authority
Builtin Object Token
Verisign Class 1 Public Primary Certification Authority
Builtin Object Token
Verisign Class 2 Public Primary Certification Authority
Builtin Object Token
Verisign Class 1 Public Primary Certification Authority - G2
Builtin Object Token
Verisign Class 2 Public Primary Certification Authority - G2
Builtin Object Token
Verisign Class 3 Public Primary Certification Authority - G2
Builtin Object Token
Verisign Class 4 Public Primary Certification Authority - G2
Builtin Object Token
VeriSign Class 1 Public Primary Certification Authority - G3
Builtin Object Token
VeriSign Class 2 Public Primary Certification Authority - G3
Builtin Object Token
VeriSign Class 3 Public Primary Certification Authority - G3
Builtin Object Token
VeriSign Class 4 Public Primary Certification Authority - G3
Builtin Object Token
Class 1 Public Primary OCSP Responder
Builtin Object Token
Class 2 Public Primary OCSP Responder
Builtin Object Token
Class 3 Public Primary OCSP Responder
Builtin Object Token
VeriSign Time Stamping Authority CA
Builtin Object Token
beTRUSTed
beTRUSTed Root CAs
Builtin Object Token
beTRUSTed Root CA-Baltimore Implementation
Builtin Object Token
beTRUSTed Root CA - Entrust Implementation
Builtin Object Token
beTRUSTed Root CA - RSA Implementation
Builtin Object Token
View
Edit
Import
Delete
OK

Certificates

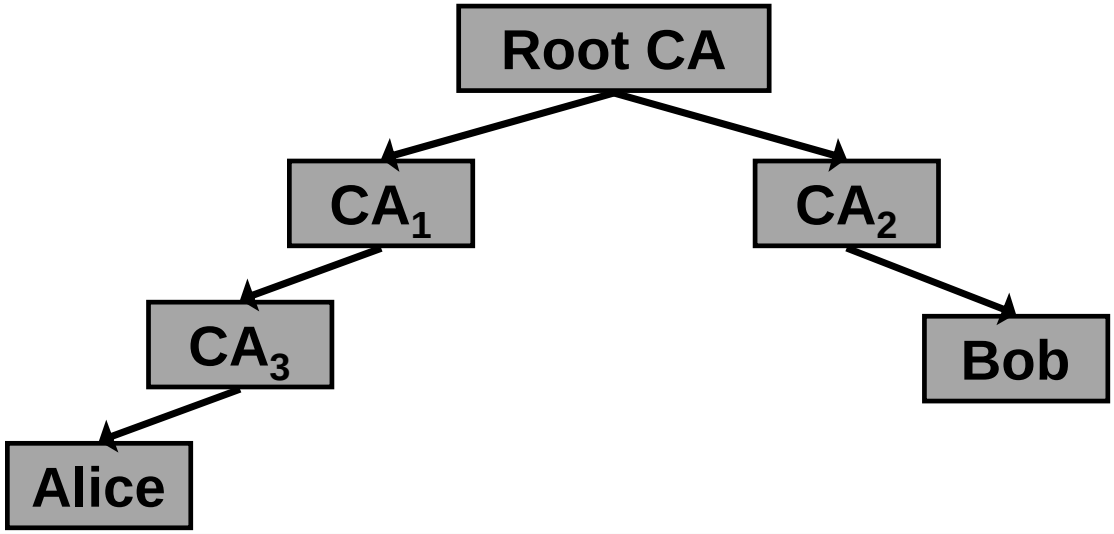
- A certificate usually contains the following information
  - Owner's name
  - Owner's public key
  - Encryption/signature algorithm
  - Name of the CA
  - Serial number of the certificate
  - Expiry date of the certificate
  - …
- Your web browser contains the public keys of some CAs
- A web site identifies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

Public Key Infrastructure (PKI)

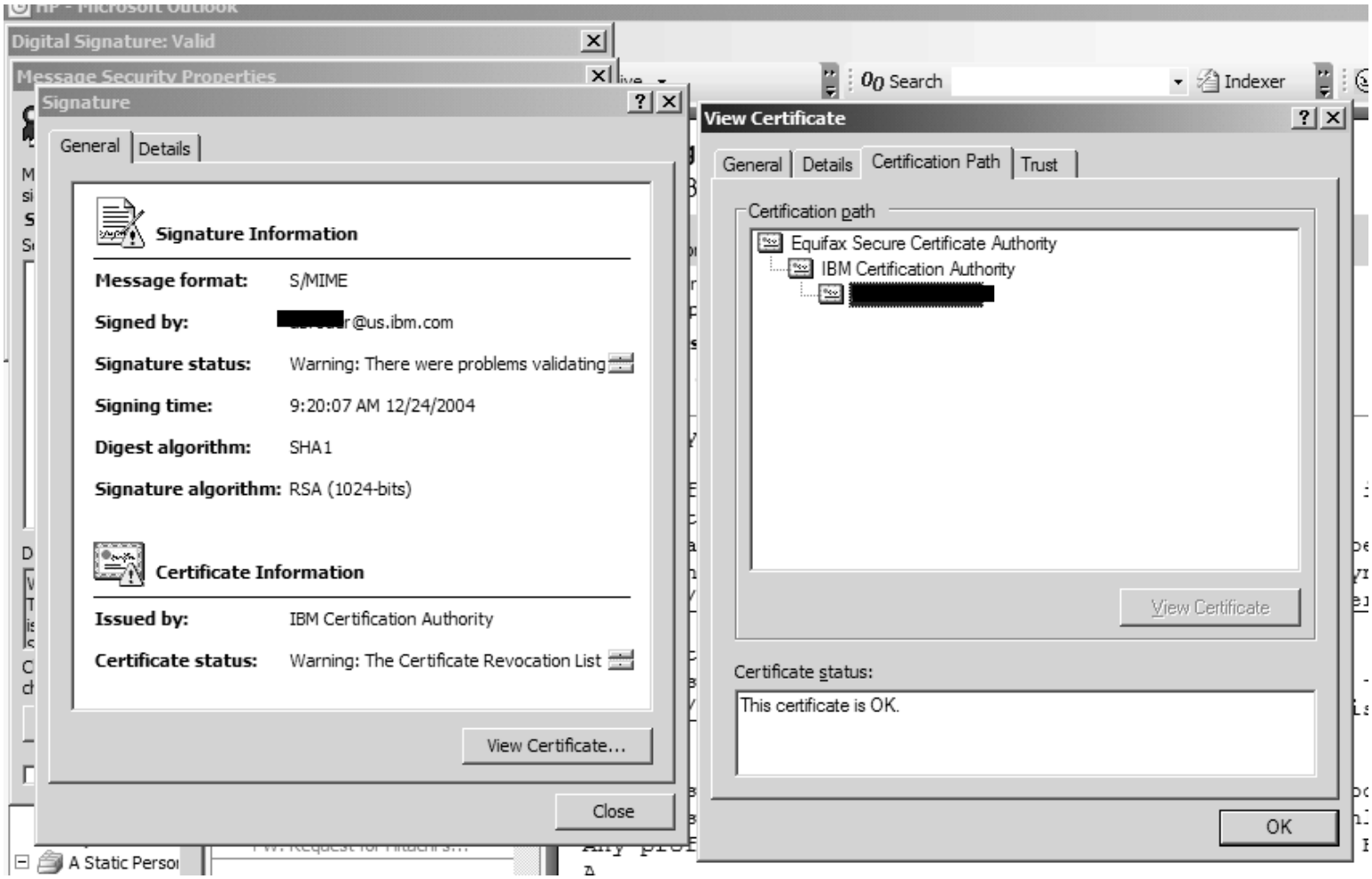
- The goal: build trust on a global level
- Running a CA:
  - If people trust you to vouch for other parties, everyone needs you.
  - A license to print money
  - But,
    - The CA should limit its responsibilities, buy insurance…
    - It should maintain a high level of security
    - Bootstrapping: 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



Public Key Infrastructure

- Oligarchy
  - Multiple trust anchors (top level CAs)
    - Pre-configured in software
    - User can add/remove CAs
- Top-down with name constraints
  - Like monopoly + delegated CAs
  - But every delegated CA has a predefined portion of the name space (il, ac.il, haifa.ac.il, cs.haifa.ac.il)
  - More trustworthy

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?

Certificate Revocation Lists (CRLs)

- A revocation agency (RA) issues a list of revoked certificates (i.e., “bad” certificates)
  - The list is updated and published regularly (e.g. daily)
  - Before trusting a certificate, users must consult the most recent CRL in addition to checking the expiry date.
- Advantages: simple.
- Drawbacks:
  - Scalability. CRLs can be huge. There is no short proof that a certificate is valid.
  - There is a vulnerability windows between a compromise of certificate and the next publication of a CRL.
  - Need a reliable way of distributing CRLs.
- Improving scalability using “delta CRLs”: a CRL that only lists certificates which were revoked since the issuance of a specific, previously issued CRL.

Explicit revocation: OCSP

- OCSP (Online Certificate Status Protocol)
  - RFC 2560, June 1999.
- OCSP can be used in place, or in addition, to CRLs
- Clients send a request for certificate status information.
  - An OCSP server sends back a response of "current", "expired," or "unknown“.
  - The response is signed (by the CA, or a Trusted Responder, or an Authorized Responder certified by the CA).
- Provides instantaneous status of certificates
  - Overcomes the chief limitation of CRL: the fact that updates must be frequently downloaded and parsed by clients to keep the list current

Certificate Revocation System (CRS)

- Certificate Revocation System (Micali'96)
- *Puts the burden of proof on the certificate holder* (who must prove that the certificate is still valid).

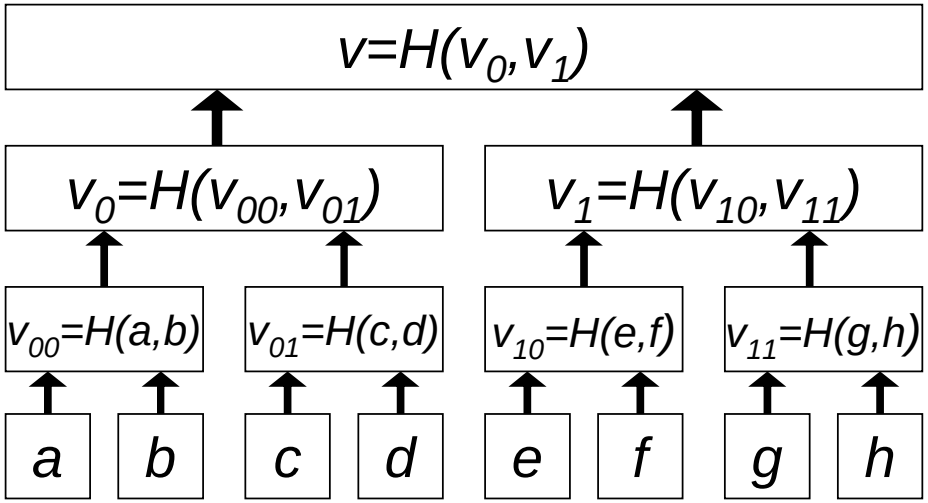
- In theory, we could limit the lifetime of certificates to a single day, and require the certificate holder to ask for a new certificate every day.
  - This would result in a high overhead at the CA

Certificate Revocation System (CRS)

- It is possible to reduce the overhead of the CA by using a hash chain
  - The certificate includes $Y_{365} = f^{365}(Y_0)$. This value is part of the information signed by the CA. f is one-way.
  - On day d,
    - If the certificate is valid, then $Y_{365-d} = f^{365-d}(Y_0)$ is sent by the CA to the certificate holder or to a directory.
    - The certificate receiver uses the daily value ($f^{365-d}(Y_0)$) to verify that the certificate is still valid. (how?)
- Advantage: A short, individual, proof per certificate.
- Disadvantage: Daily overhead, even when a cert is valid.

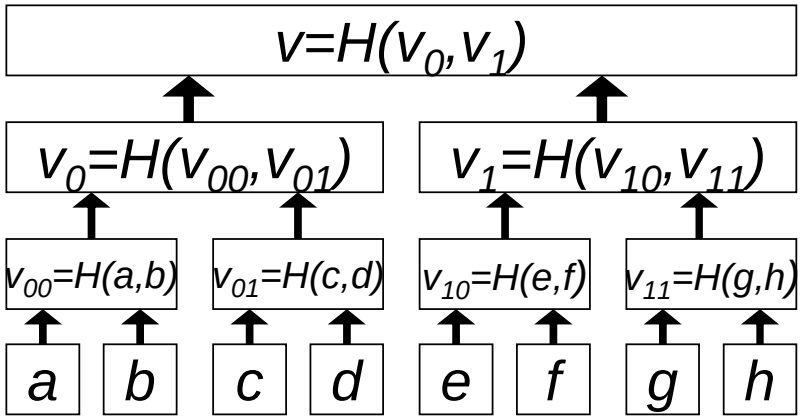
Merkle Hash Tree (will be useful later)

- A method of committing to (by hashing together) *n* values, $x_1,\ldots,x_n$, such that
  - The result is a single hash value
  - For any x_i, it is possible to prove that it appeared in the original list, using a proof of length O(log n).



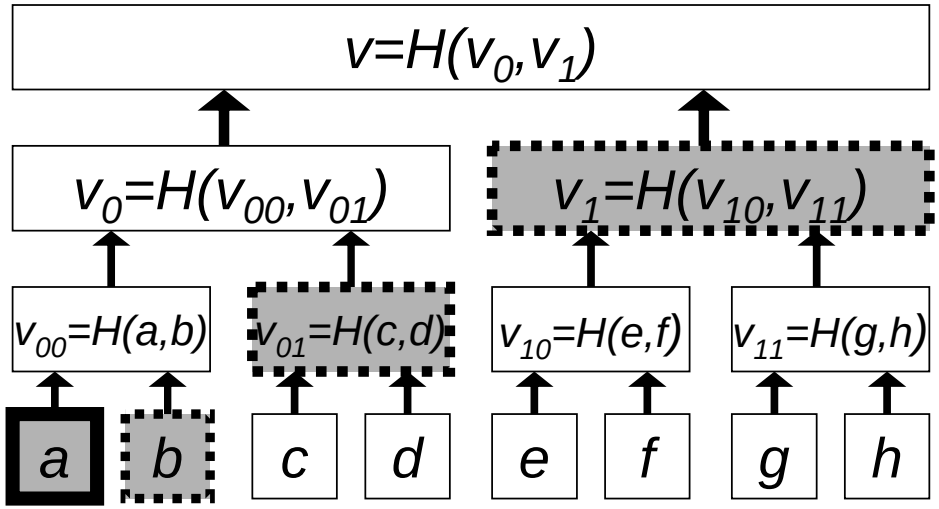
Merkle Hash Tree

- H is a collision intractable hash function
- Any change to a leaf results in a change to the root
- To sign the set of values it is sufficient to sign the root (a single signature instead of *n*).
- How do we verify that an element appeared in the signed set?



Verifying that *a* appears in the signed set

- Provide *a*'s leaf, and the siblings of the nodes in the path from *a* to the root. (O(log n) values)
- The verifier can use *H* to compute the values of the nodes in the path from the leaf to the root.
- It then compares the computed root to the signed value.



Using hash trees to improve the overhead of CRS

- Originally (for a year long certificate)
  - the certificate includes $f^{365}(Y_0)$
  - On day d, certificate holder obtains $f^{365-d}(Y_0)$
  - The certificate receiver computes $f^{365}(Y_0)$ from $f^{365-d}(Y_0)$ by invoking $f()$ d times.
- Slight improvement:
  - The CA assigns a different leaf for every day, constructs a hash tree, and signs the root.
  - On day d, it releases node d and the siblings of the path from it to the root.
  - This is the proof that the certificate is valid on day d
  - The overhead of verification is O(log 365).

Certificate Revocation Tree (CRT) [Kocher]

- (A different usage of a hash tree)
- A CRT is a hash tree with leaves corresponding to statements about ranges of certificates
  - Statements describe regions of certificate ids, in which only the smallest id is revoked.
    - For example, a leaf might read: “if 100 ≤ id <234, then cert is revoked iff id=100”.
  - Each certificate matches exactly one statement.
  - The statements are the leaves of a signed hash tree, ordered according to the ranges of certificate values.
  - To examine the state of a certificate we retrieve the statement for the corresponding region.
  - A single hash tree is used for all certs.

Certificate Revocation Tree (CRT)

- Preferred operation mode:
  - Every day the CA constructs an updated tree.
  - The CA signs a statement including the root of the tree and the date.
  - It is Alice's responsibility to retrieve the leaf which shows that her certificate is valid, the route from this leaf to the root, and the CA's signature of the root.
  - To prove the validity of her cert, Alice sends this information.
  - The receiver verifies the value in the leaf, the route to the tree, and the signature.
- Advantage:
  - a short proof for the status of a certificate.
  - The CA does not have to handle individual requests.
- Drawback: the entire hash tree must be updated daily.

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

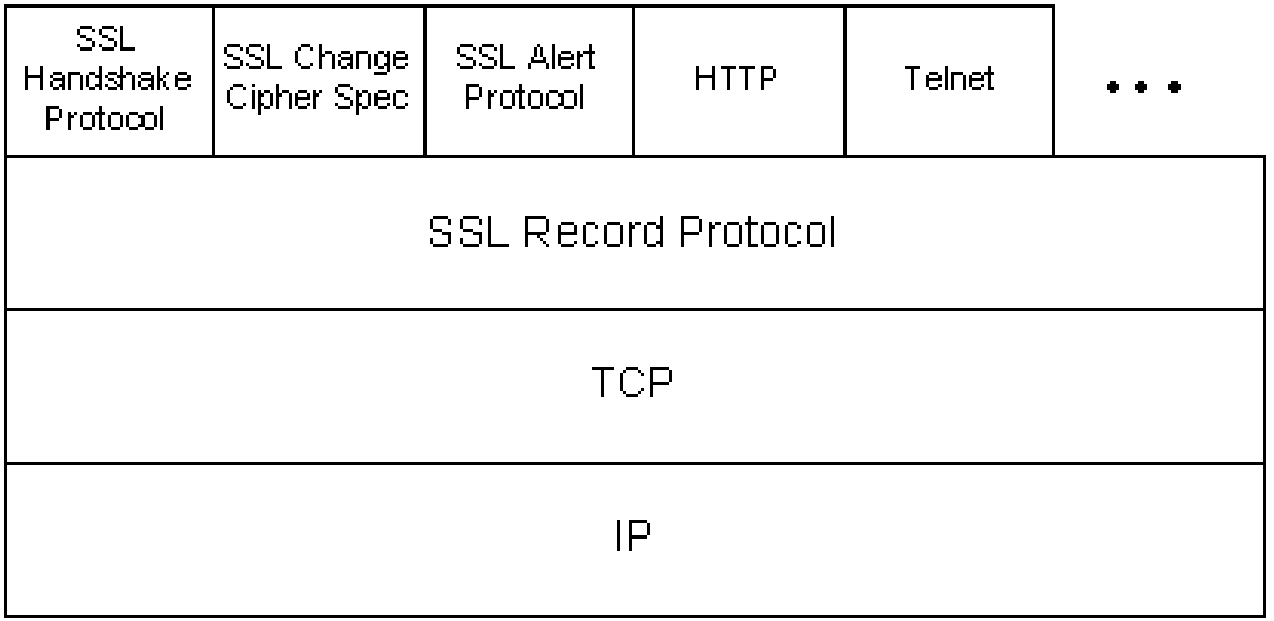
(1) This is a very simplified version of the actual protocol.

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

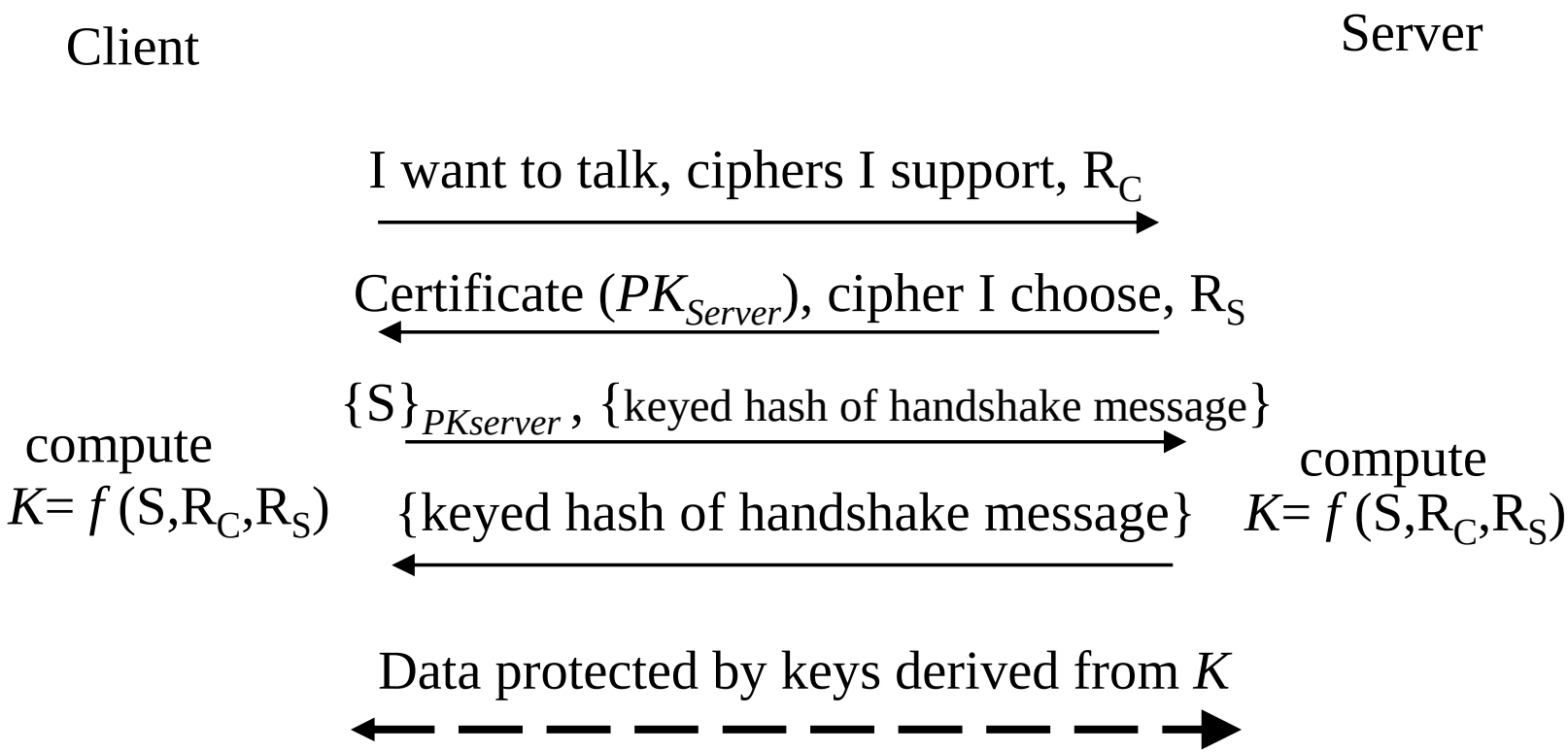
- SSL/TLS operates over TCP, which ensures reliable transport.
- Supports any application protocol (usually used with http).



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



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

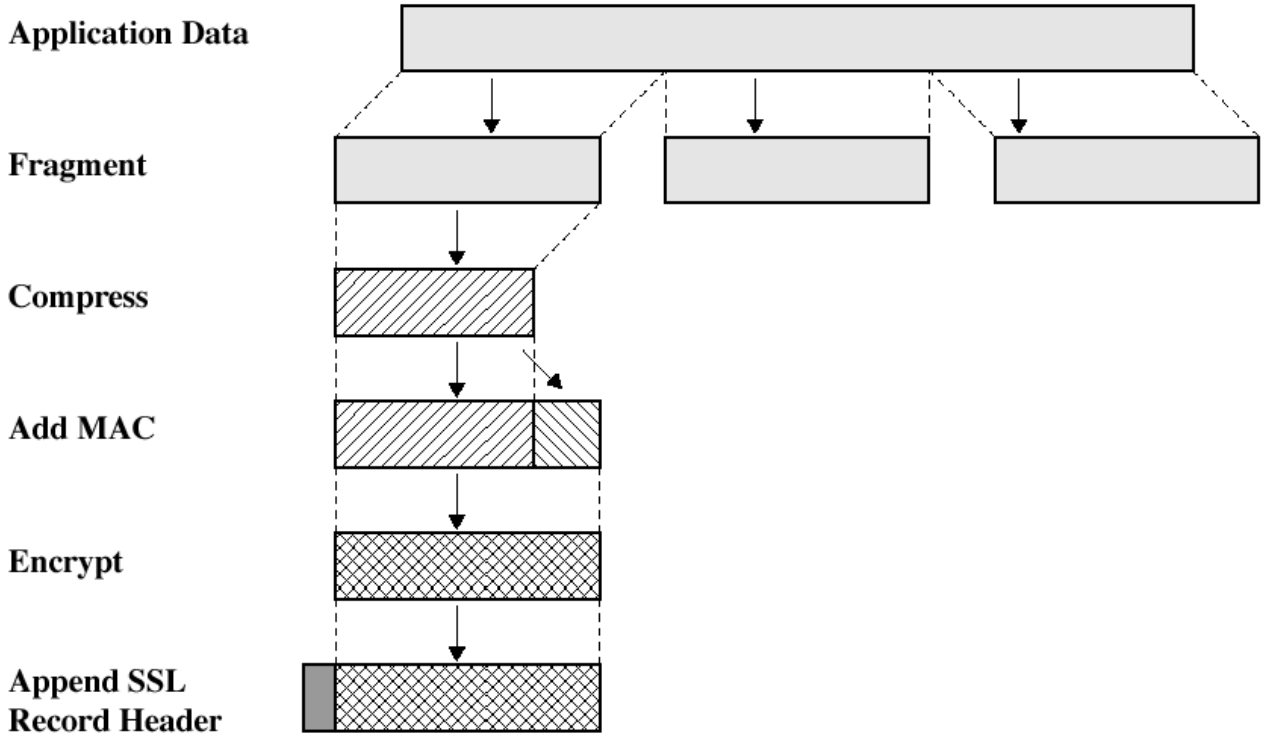


Figure 17.3 SSL Record Protocol Operation