Introduction to Cryptography Lecture 10

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

Benny Pinkas

Certification Authorities (CA)

- 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)

- 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)*.
- 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 interaction with the CA.

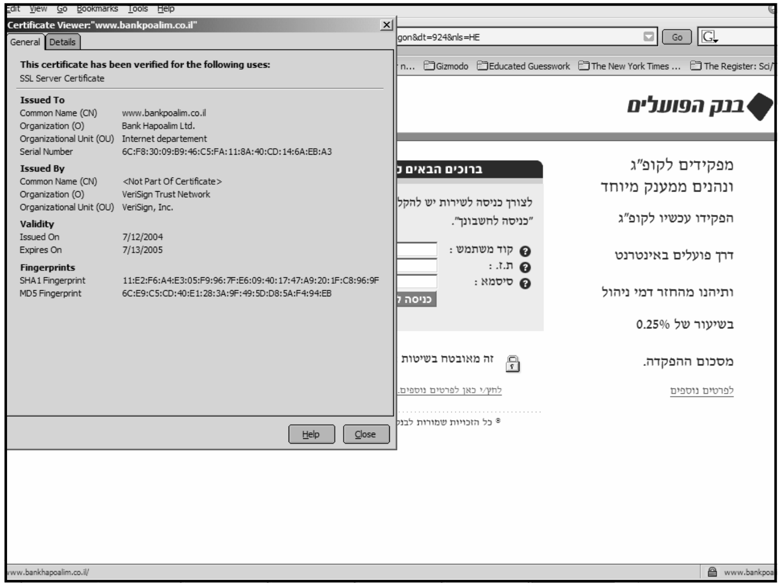
Certification Authorities (CA)

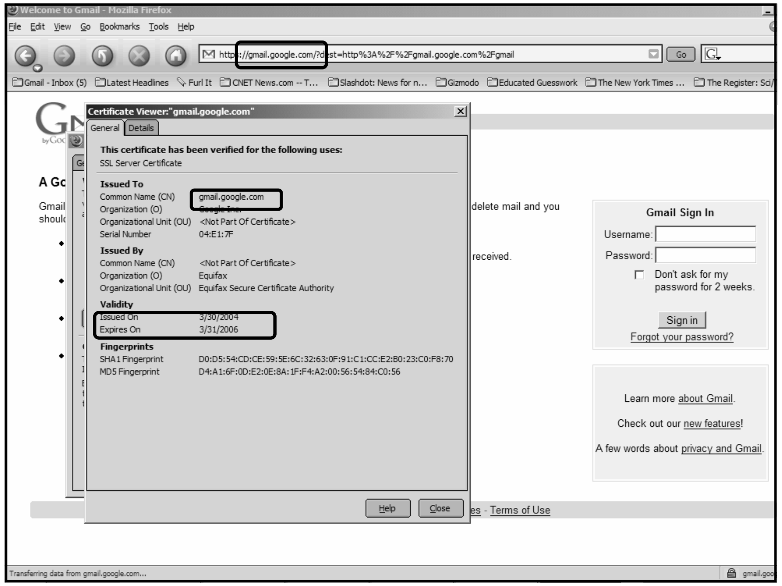
- Unlike KDCs, the CA does not have to be online to provide keys to users
  - It can therefore be better secured than a KDC
  - The CA does not have to be available all the time
- Users only keep a single public key – of the CA
- The certificates are not secret. They can be stored in a public place.
- 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.

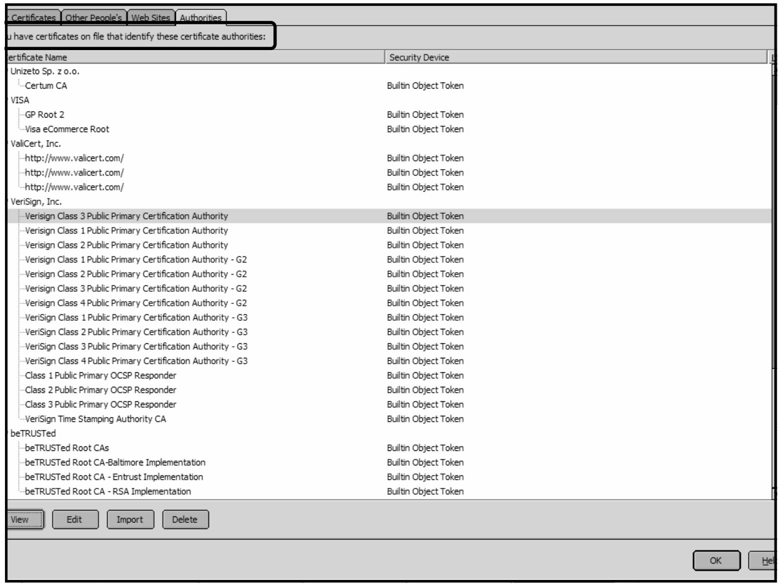
Certification Authorities (CA)

- For 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.







Certificates

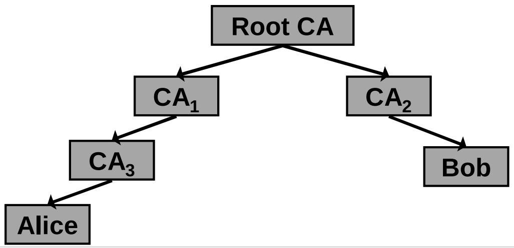
- A certificate usually contains the following information
  - Owner's name
  - Owner's public key
  - Encryption/signature algorithm
  - Name of the CA
  - Serial number of the certificate
  - Expiry date of the certificate
  - …
- Your web browser contains the public keys of some CAs
- A web site identifies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

Public Key Infrastructure (PKI)

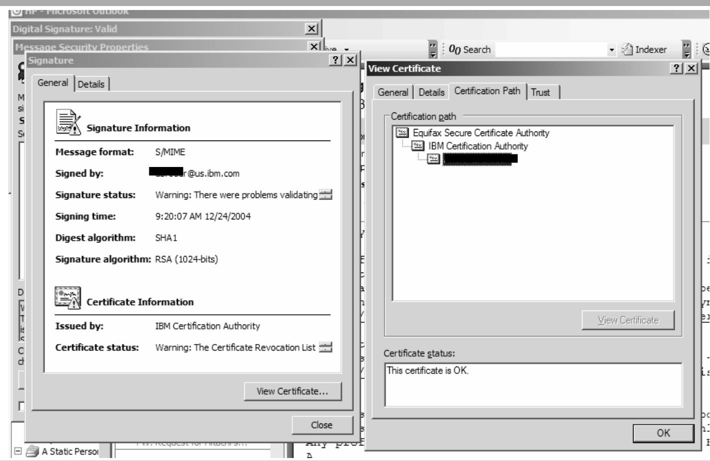
- The goal: build trust on a global level
- Running a CA:
  - If people trust you to vouch for other parties, everyone needs you.
  - A license to print money
  - But,
    - The CA should limit its responsibilities, buy insurance…
    - It should maintain a high level of security
    - Bootstrapping: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

Public Key Infrastructure (PKI)

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



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

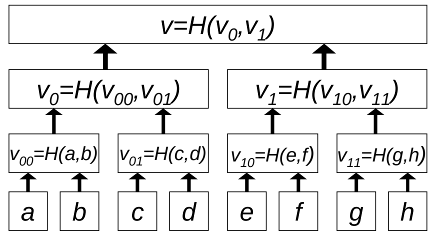
- OCSP (Online Certificate Status Protocol)
  - RFC 2560, June 1999.
- OCSP can be used in place, or in addition, to CRLs
- Clients send a request for certificate status information.
  - An OCSP server sends back a response of "current", "expired," or "unknown“.
  - The response is signed (by the CA, 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*
- Uses 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.

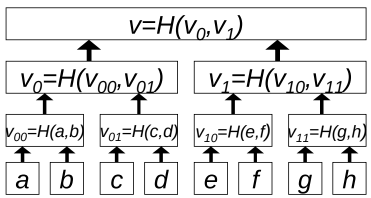
Merkle Hash Tree

- 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).



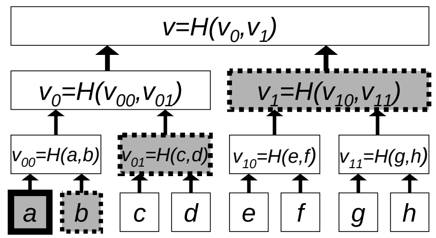
Merkle Hash Tree

- 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 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.

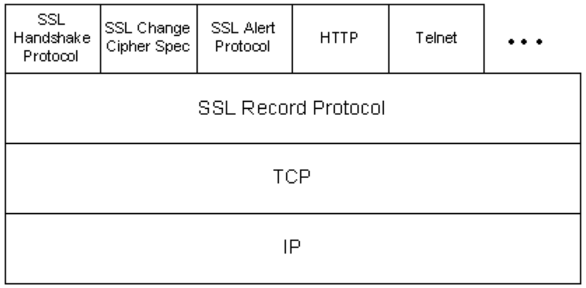
SSL/TLS

- 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

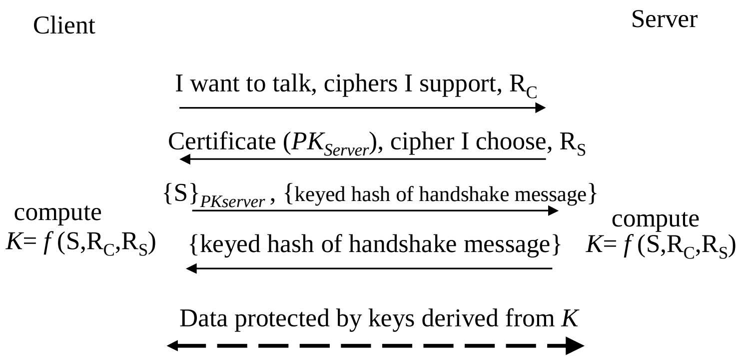
- Handshake Protocol - establishes a session
  - Agreement on algorithms and security parameters
  - Identity authentication
  - Agreement on a key
  - Report error conditions to each other

- 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 ⇒ 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 ⇒ 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 ⇒ 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

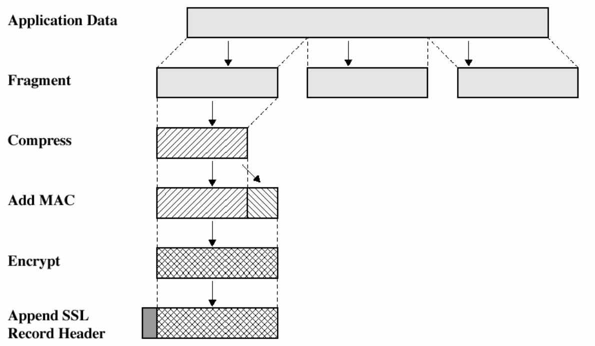


Figure 17.3 SSL Record Protocol Operation