

Introduction to Cryptography

Lecture 9

Digital signatures,
Public Key Infrastructure (PKI)

Benny Pinkas

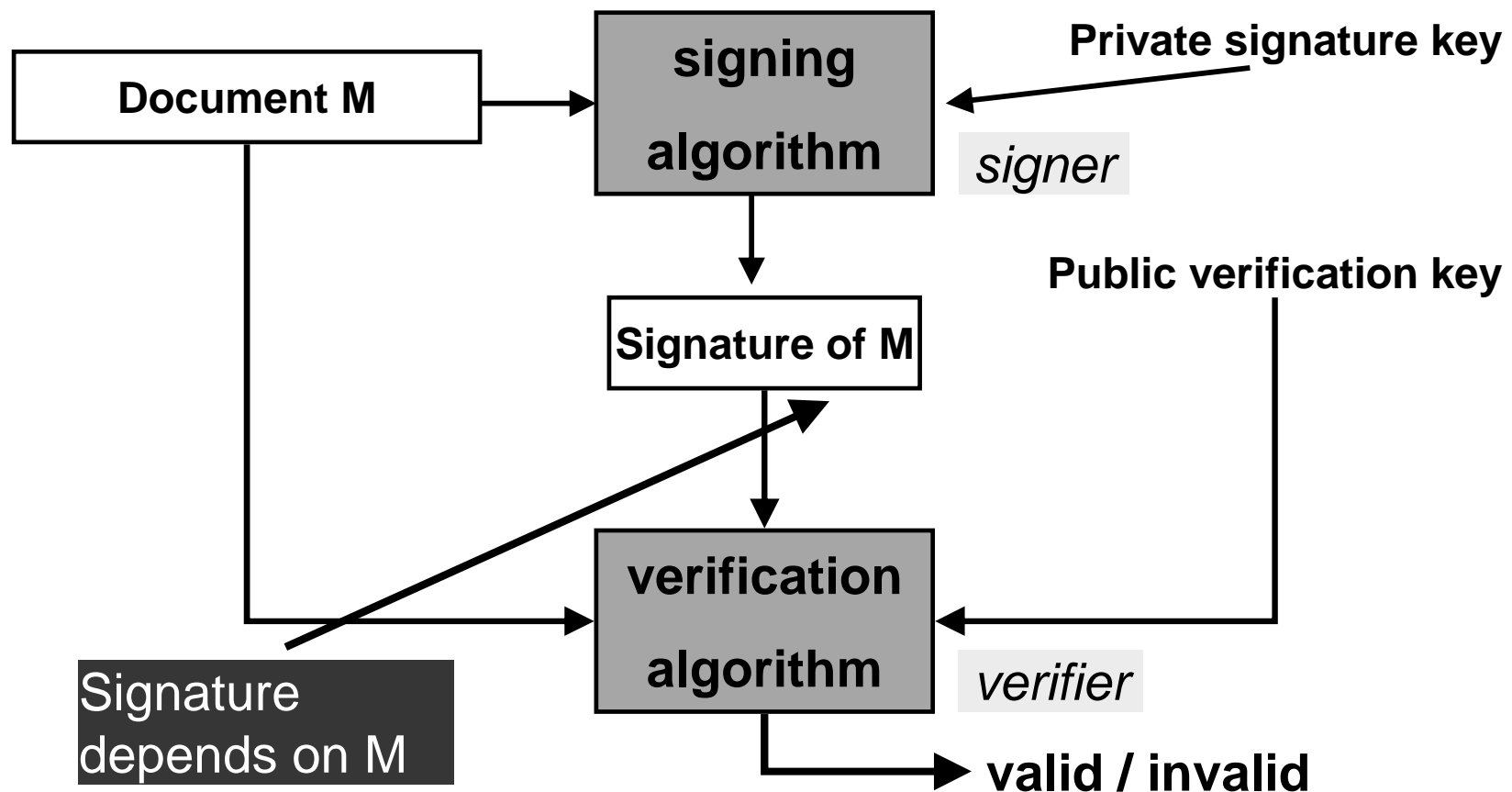
Desiderata for digital signatures

- Associate a document to an signer
- A digital signature is attached to a document (*rather than be part of it*)
- The signature is easy to verify but hard to forge
 - Signing is done using knowledge of a private key
 - Verification is done using a public key associated with the signer (*rather than comparing to an original signature*)
 - It is impossible to change even one bit in the signed document
- *A copy of a digitally signed document is as good as the original signed document.*
- Digital signatures could be legally binding...

Non Repudiation

- Prevent signer from denying that it signed the message
- I.e., the receiver can prove to third parties that the message was signed by the signer
- This is different than message authentication (MACs)
 - There the receiver is assured that the message was sent by the receiver and was not changed in transit
 - But the receiver cannot prove this to other parties
 - MACs: sender and receiver share a secret key K
 - If R sees a message MACed with K , it knows that it could have only been generated by S
 - But if R shows the MAC to a third party, it cannot prove that the MAC was generated by S and not by R

Signing/verification process



Diffie-Hellman

“New directions in cryptography” (1976)

- In public key encryption
 - The encryption function is a trapdoor permutation f
 - Everyone can encrypt = compute $f()$. (using the public key)
 - Only Alice can decrypt = compute $f^{-1}()$. (using her private key)
- Alice can use f for signing
 - Alice signs m by computing $s=f^{-1}(m)$.
 - Verification is done by computing $m=f(s)$.
- Intuition: since only Alice can compute $f^{-1}()$, forgery is infeasible.
- Caveat: none of the established practical signature schemes following this paradigm is provably secure

Example: simple RSA based signatures

- Key generation: (as in RSA)
 - Alice picks random p, q . Finds $e \cdot d = 1 \bmod (p-1)(q-1)$.
 - Public verification key: (N, e)
 - Private signature key: d
- Signing: Given m , Alice computes $s = m^d \bmod N$.
- Verification: given m, s and public key (N, e) .
 - Compute $m' = s^e \bmod N$.
 - Output “valid” iff $m' = m$.

Message lengths

- A technical problem:
 - $|m|$ might be longer than $|N|$
 - m might not be in the domain of $f^{-1}()$

Solution:

- Signing: First compute $H(m)$, then compute the signature $f^{-1}(H(m))$. Where,
 - $H()$ must be collision intractable. I.e. it is hard to find m, m' s.t. $H(m)=H(m')$.
 - The range of $H()$ must be contained in the domain of $f^{-1}()$.
- Verification:
 - Compute $f(s)$. Compare to $H(m)$.
- Use of $H()$ is also good for security reasons. See below.

Security of using hash function

- Intuitively
 - Adversary can compute $H()$, $f()$, but not $H^{-1}()$, $f^{-1}()$.
 - Can only compute $(m, H(m))$ by choosing m and computing $H()$.
 - Adversary wants to compute $(m, f^{-1}(H(m)))$.
 - To break signature needs to show s s.t. $f(s)=H(m)$. (E.g. $s^e=H(m)$.)
 - Failed attack strategy 1:
 - Pick s , compute $f(s)$, and look for m s.t. $H(m)=f(s)$.
 - Failed attack strategy 2:
 - Pick m, m' s.t. $H(m)=H(m')$. Ask for a signature s of m' (which is also a signature of m).
 - (If $H()$ is not collision resistant, adversary could find m, m' s.t. $H(m) = H(m')$.)
 - This doesn't mean that the scheme is secure, only that these attacks fail.

Security definitions for digital signatures

- Attacks against digital signatures
 - *Key only attack*: the adversary knows only the verification key
 - *Known signature attack*: in addition, the adversary has some message/signature pairs.
 - *Chosen message attack*: the adversary can ask for signatures of messages of its choice (e.g. attacking a notary system).
(Seems even more reasonable than chosen message attacks against encryption.)

Security definitions for digital signatures

- Several levels of success for the adversary
 - *Existential forgery*: the adversary succeeds in forging the signature of one message.
 - *Selective forgery*: the adversary succeeds in forging the signature of one message of its choice.
 - *Universal forgery*: the adversary can forge the signature of any message.
 - *Total break*: the adversary finds the private signature key.
- Different levels of security, against different attacks, are required for different scenarios.

Example: simple RSA based signatures

- Key generation: (as in RSA)
 - Alice picks random p, q . Defines $N=pq$ and finds $e \cdot d = 1 \bmod (p-1)(q-1)$.
 - Public verification key: (N, e)
 - Private signature key: d
- Signing: Given m , Alice computes $s = m^d \bmod N$.
- *(suppose that there is no hash function $H()$)*
- Verification: given m, s and public key (N, e) .
 - Compute $m' = s^e \bmod N$.
 - Output “valid” iff $m' = m$.

Attacks against plain RSA signatures

- Signature of m is $s = m^d \bmod N$.
- Universally forgeable under a chosen message attack:
 - *Universal forgery*: the adversary can forge the signature of any message of its choice.
 - *Chosen message attack*: the adversary can ask for signatures of messages of its choice.
- Existentially forgeable under key only attack.
 - *Existential forgery*: succeeds in forging the signature of at least one message.
 - *Key only attack*: the adversary knows the public verification key but does not ask any queries.

RSA with a full domain hash function

- Signature is $\text{sig}(m) = f^{-1}(H(m)) = (H(m))^d \bmod N$.
 - $H()$ is such that its range is $[1, N]$
- *The system is no longer homomorphic*
 - $\text{sig}(m) \cdot \text{sig}(m') \neq \text{sig}(m \cdot m')$
- *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

RSA with full domain hash –proof of security

- Claim: Assume that $H()$ is a random function, then if there is a polynomial-time $A()$ which forges a signature with non-negligible probability, then it is possible to invert the RSA function, on a random input, with non-negligible probability.
- Proof:
 - Our input: y . Should compute $y^d \bmod N$.
 - $A()$ queries $H()$ and a signature oracle $\text{sig}()$, and generates a signature s of a message for which it did not query $\text{sig}()$.
 - Suppose $A()$ made at most t queries to $H()$, asking for $H(m_1), \dots, H(m_t)$. Suppose also that it always queries $H(m)$ before querying $\text{sig}(m)$.
 - We will show how to use $A()$ to compute $y^d \bmod N$.

RSA with full domain hash –proof of security

- Proof (contd.)
 - We can decide how to answer A 's queries to $H(), \text{sig}()$.
 - Choose a random i in $[1, t]$, answer queries to $H()$ as follows:
 - The answer to the i th query (m_i) is y .
 - The answer to the j th query ($j \neq i$) is $(r_j)^e$, where r_j is random.
 - Answer to $\text{sig}(m)$ queries:
 - If $m = m_j$, $j \neq i$, then answer with r_j . (Indeed $\text{sig}(m_j) = (H(m_j))^d = r_j$)
 - If $m = m_i$ then stop. (we failed)
 - A 's output is (m, s) .
 - If $m = m_i$ and s is the correct signature, then we found y^d .
 - Otherwise we failed.
 - Success probability is $1/t$ times success probability of $A()$.

Rabin signatures

- Same paradigm:
 - $f(m) = m^2 \bmod N$. ($N=pq$).
 - $\text{Sig}(m) = s$, 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 $\frac{1}{4}$ 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.

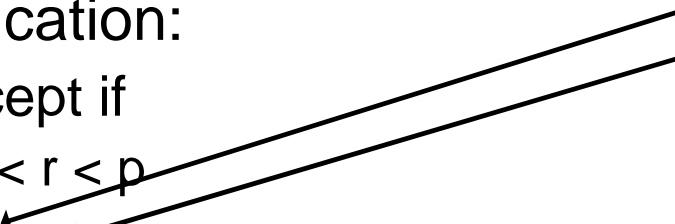
El Gamal signatures

- 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
 - $0 < r < p$
 - $y^r \cdot r^s \equiv g^m \bmod p$
- 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.

same r in
both places!



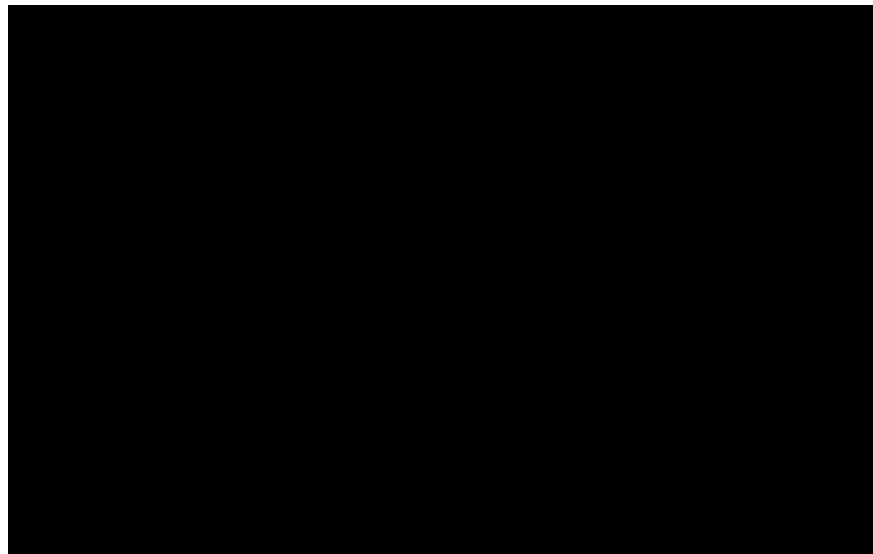
El Gamal signature: comments

- Can work in any finite Abelian group
 - The discrete log problem appears to be harder in elliptic curves over finite fields than in \mathbb{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 (PKI)

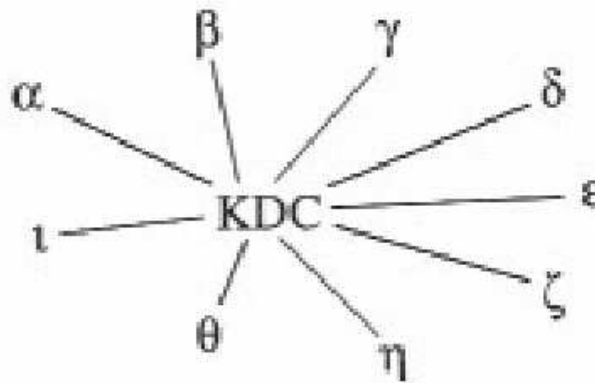
Key Infrastructure for symmetric key encryption

- 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*
 - generates 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

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.

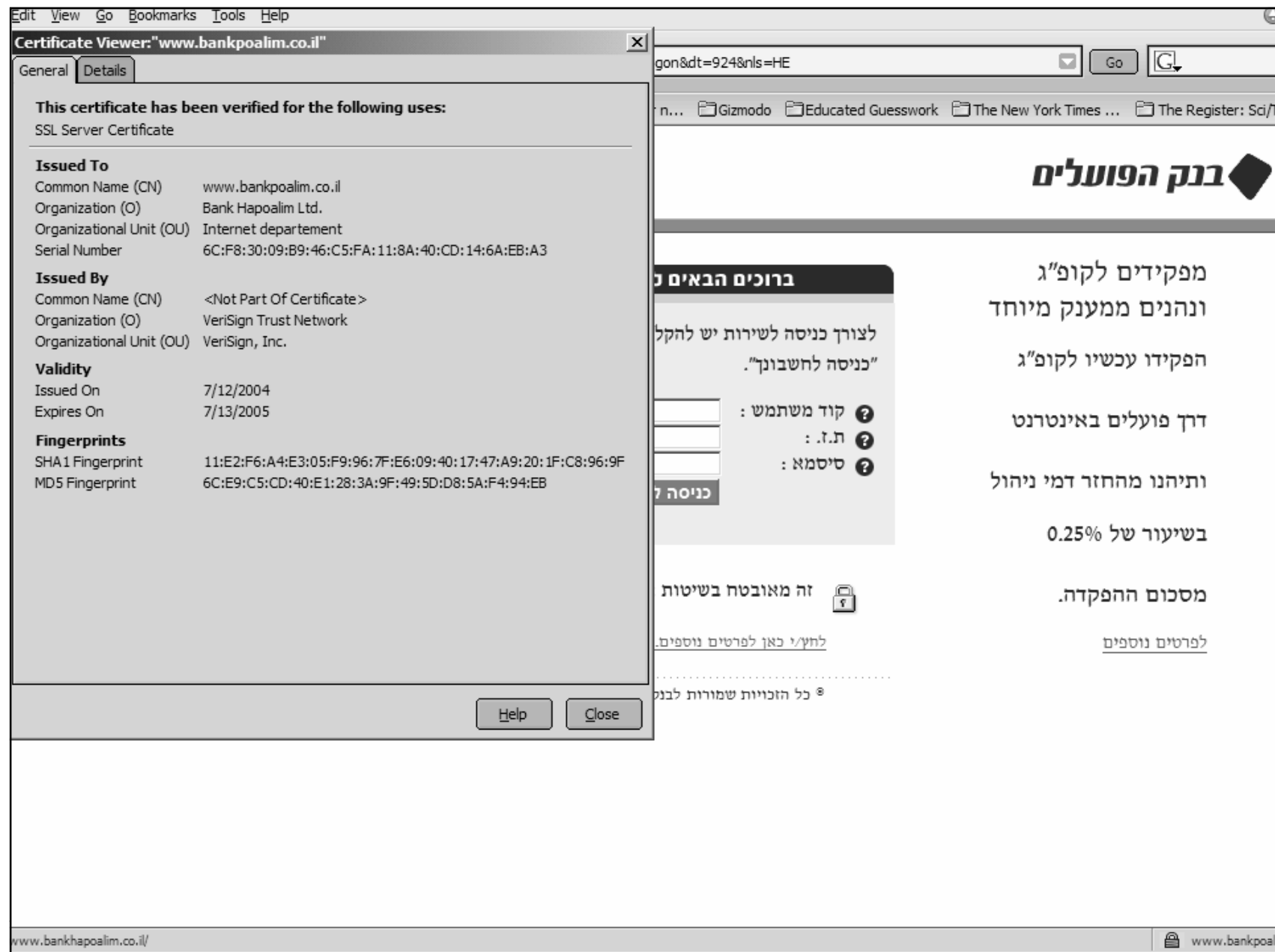
Certification Authorities (CA)

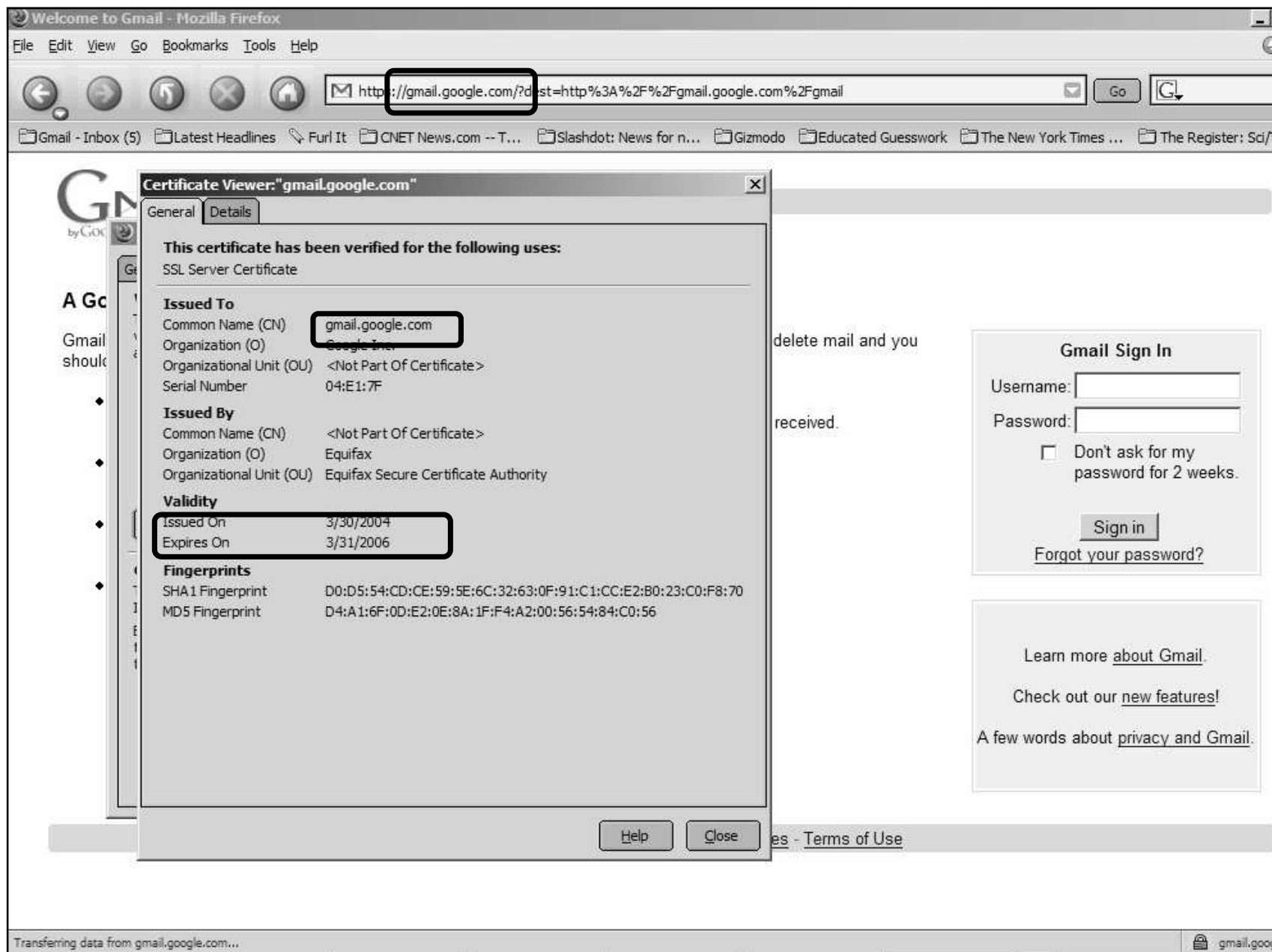
- Unlike KDCs, the CA does not have to be online to provide keys to users
 - It can therefore be better secured than a KDC
 - The CA does not have to be available all the time
- Users only keep a single public key – of the CA
- The certificates are not secret. They can be stored in a public place.
-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

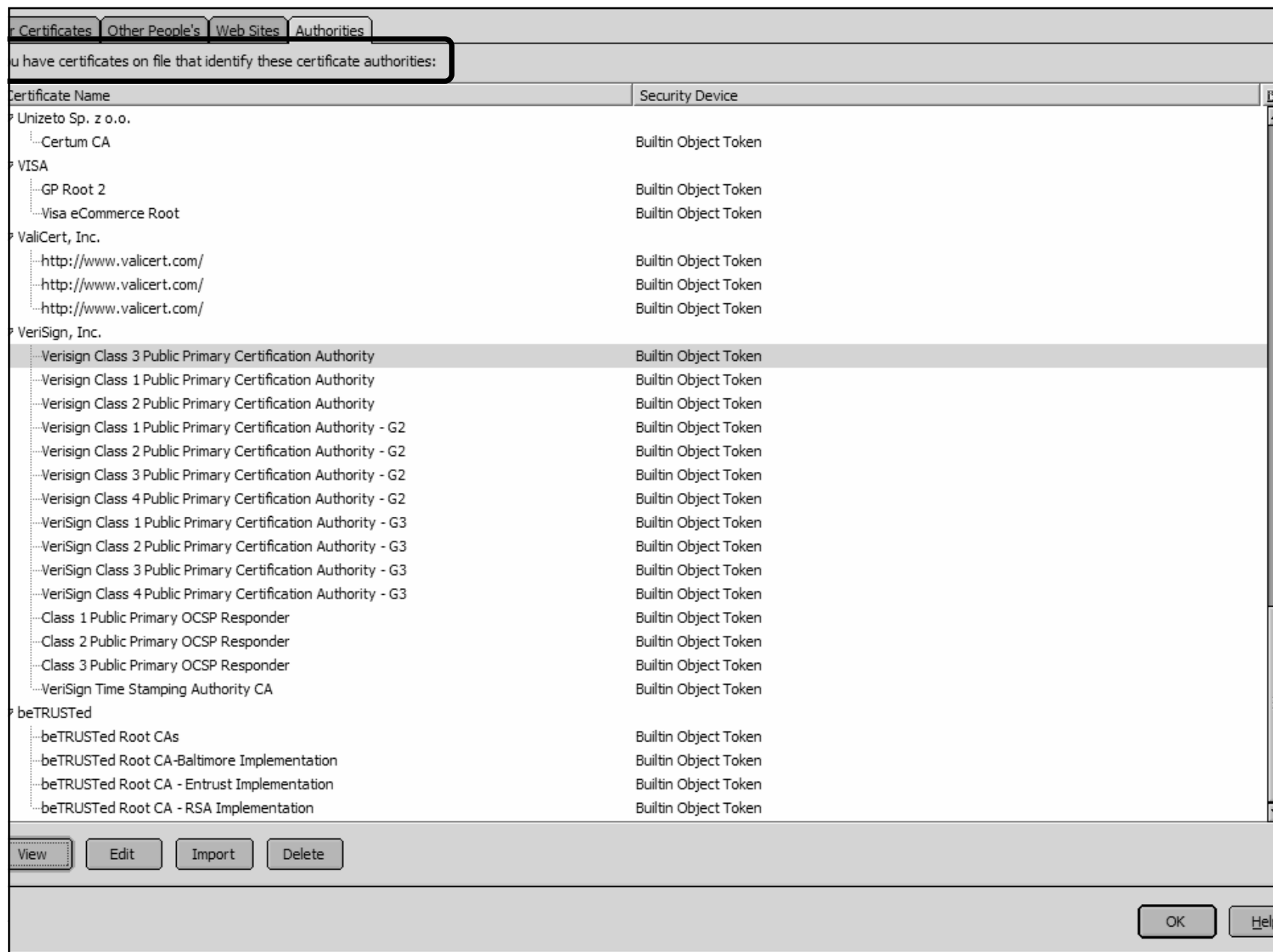
Certification Authorities (CA)

- 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

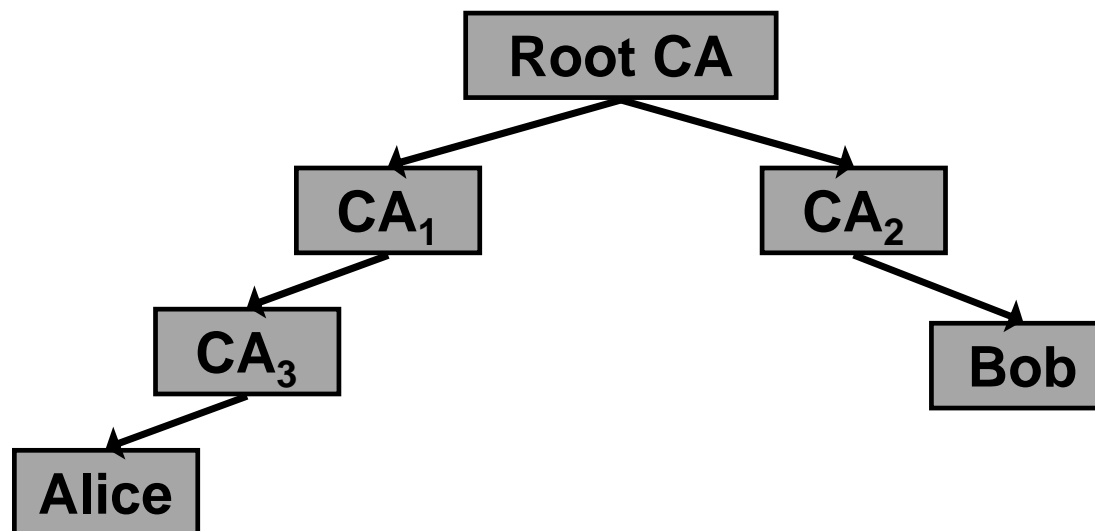
- A certificate usually contains the following information
 - Owner's name
 - Owner's public key
 - Encryption/signature algorithm
 - Name of the CA
 - Serial number of the certificate
 - Expiry date of the certificate
 - ...
- Your web browser contains the public keys of some CAs
- A web site identifies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

Public Key Infrastructure (PKI)

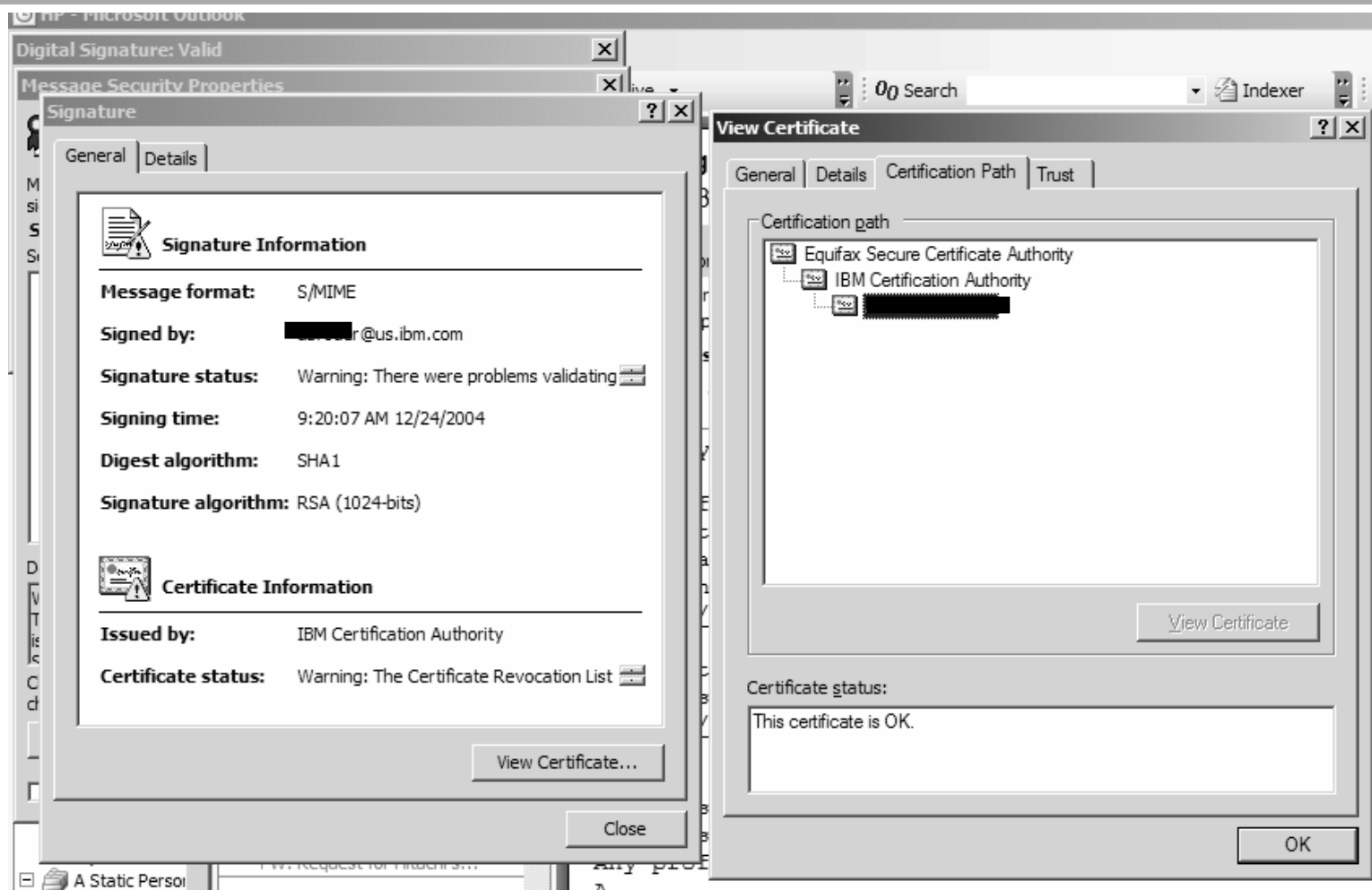
- The goal: build trust on a global level
- Running a CA:
 - If people trust you to vouch for other parties, everyone needs you.
 - A license to print money
 - But,
 - The CA should limit its responsibilities, buy insurance...
 - It should maintain a high level of security
 - Bootstrapping: how would 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
 - $[(Alice, PK_A)_{CA3}, (CA3, PK_{CA3})_{CA1}, (CA1, PK_{CA1})_{TOP-CA}]$



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