Introduction to Cryptography
Lecture 8

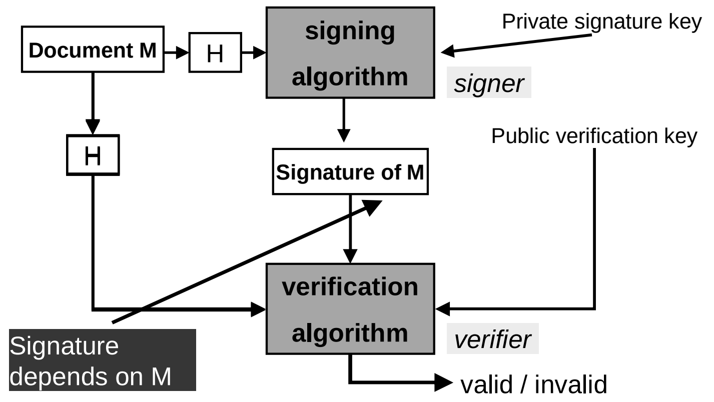
Digital signatures,
Public Key Infrastructure (PKI)

Benny Pinkas

Desiderata for digital signatures

- Associate a document to an signer
- A digital signature is attached to a document *(rather then 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…

Signing/verification process



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

Attacks against plain RSA signatures

- Signature of m is $s=m^d$ mod 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 will 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

RSA with full domain hash –proof of security

- Claim: If H() is a random oracle, 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-neg prob.
- Proof:
  - Our input: *y.* Should compute y^d mod *N.*
  - *A()* queries *H()* and a signature oracle *sig(),* and generates a signature *s* of a message for which it did not query *sig()*.
  - Suppose *A()* made at most *t* queries to *H()*, and always queries *H(m)* before querying *sig(m)*.
  - We will show how to use *A()* to compute y^d mod *N.*

RSA with full domain hash –proof of security

- Proof (contd.)
  - We decide how to answer *A's* queries to *H(),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 *sig(m)* queries:
    - If $m=m_j$, $j \neq i$, then answer with r_j. *(Indeed* $sig(m_j) = (H(m_j))^d = r_j$ *)*
    - 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:
  - $f(m) = m^2$ mod *N.*   *(N=pq).*
  - *Sig(m) = s, s.t.* $s^2 = m$ *mod 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 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 DSS (Digital Signature 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 $1 < a < p\text{-}1$.
  - Public key $p, g, y=g^a$.
- Signature: (of M)
  - Pick random $1 < k < p\text{-}1$, s.t. gcd$(k,p\text{-}1)=1$.
  - Compute $m=H(M)$.
    - $r = g^k \bmod p$.
    - $s = (m - r \cdot a) \cdot k^{-1} \bmod (p\text{-}1)$
  - Signature is r, s.

El Gamal signatures

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

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.

Public Key Infrastructure (PKI)

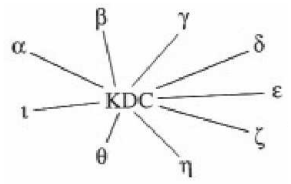
Key Infrastructure for symmetric key encryption

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

[redacted]

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 for failure, for both
    - security,
    - and 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 key
- 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.

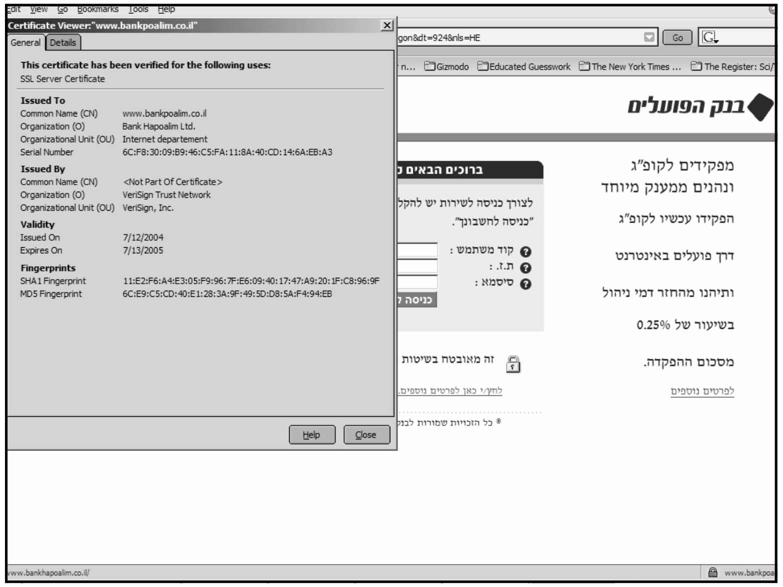
Certification Authorities (CA)

- 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.


Edit View Go Bookmarks Tools Help
General
Details
gon&dt=924&nls=HE
Go
n... Gizmodo Educated Guesswork The New York Times ... The Register: Sci/
Help
Close
www.bankhapoalim.co.il/
www.bankpoa

Certificate Viewer:"www.bankpoalim.co.il"

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

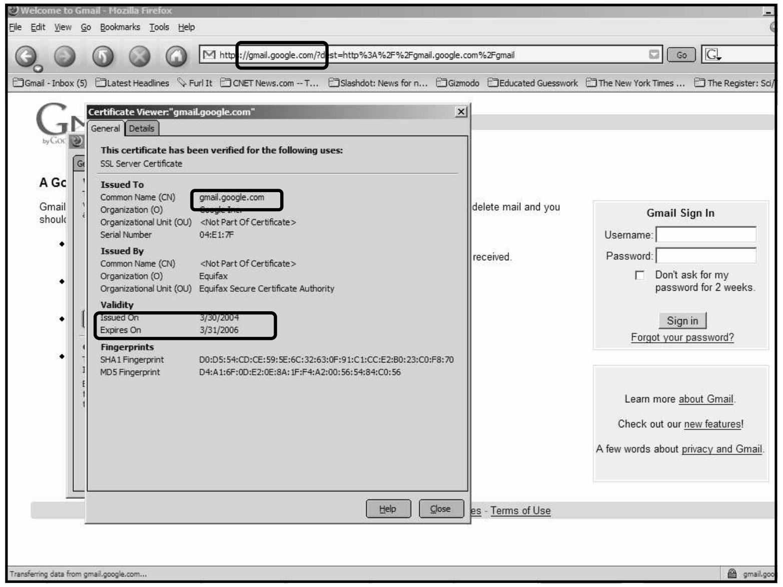
Fingerprints

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

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

ברוכים הבאים ל
לצורך כניסה לשירות יש להקל
"כניסה לחשבונך".
קוד משתמש :
ת.ז. :
סיסמא :
כניסה ל
זה מאובטח בשיטות
לחץ/י כאן לפרטים נוספים.
® כל הזכויות שמורות לבנק


Welcome to Gmail - Mozilla Firefox

http://gmail.google.com/?dest=http%3A%2F%2Fgmail.google.com%2Fgmail

Certificate Viewer:"gmail.google.com"

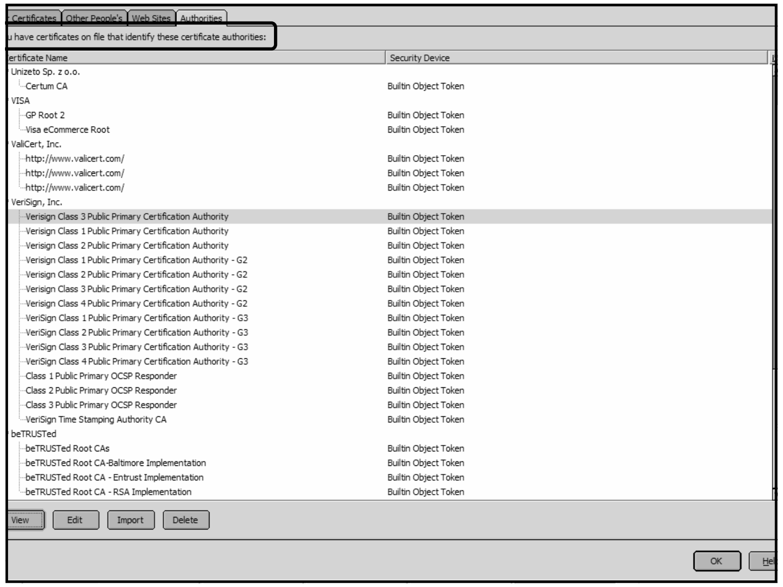
This certificate has been verified for the following uses:
SSL Server Certificate

Issued To	
Common Name (CN)	gmail.google.com
Organization (O)	Google Inc
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

Gmail Sign In
Username:
Password:
Don't ask for my password for 2 weeks.
Forgot your password?

Learn more about Gmail.
Check out our new features!
A few words about privacy and Gmail.

Transferring data from gmail.google.com...


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

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

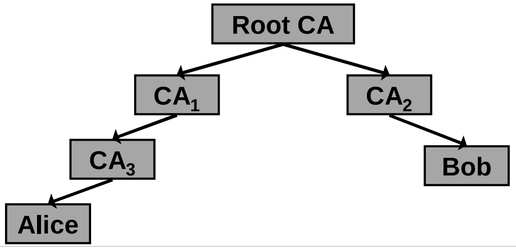
Public Key Infrastructure (PKI)

- The goal: build trust on a global level

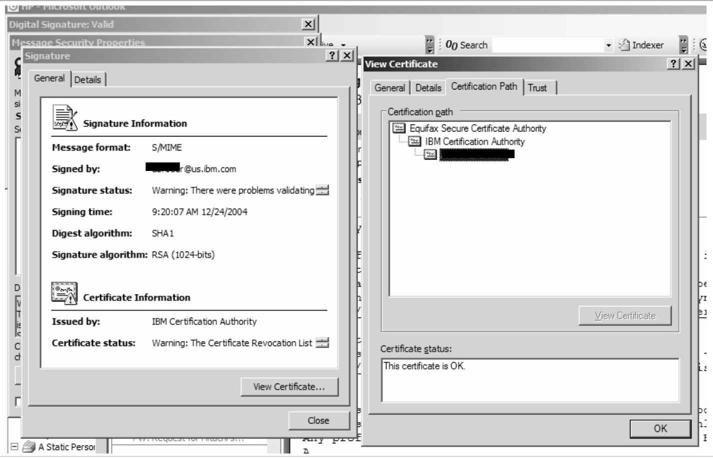
- Running a CA:
  - If people trust you to vouch for other parties, everyone needs you.
  - A license to print money
  - But,
    - The CA should limit its responsibilities, buy insurance…
    - It should maintain a high level of security
    - Bootstrapping: how would everyone get the CA's public key?

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
    - [$(\text{Alice}, PK_A)_{CA3}$, $(CA3, PK_{CA3})_{CA1}$, $(CA1, PK_{CA1})_{TOP\text{-}CA}$]



Certificate chain



Signature Information

Message format: S/MIME

Signed by: [redacted]@us.ibm.com

Signature status: Warning: There were problems validating

Signing time: 9:20:07 AM 12/24/2004

Digest algorithm: SHA1

Signature algorithm: RSA (1024-bits)

Certificate Information

Issued by: IBM Certification Authority

Certificate status: Warning: The Certificate Revocation List

Certification path

```
Equifax Secure Certificate Authority
  IBM Certification Authority
    [redacted]
```

Certificate status:

This certificate is OK.

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