IN2120 Information Security
University of Oslo
Autumn 2019

Lecture 3
## Key Management and PKI

Audun Jøsang

Key Management

- The strength of cryptographic security depends on:
  1. The size of the keys
  2. The robustness of cryptographic algorithms/protocols
  3. The protection and management afforded to the keys
- Key management provides the foundation for the secure generation, storage, distribution, and destruction of keys.
- Key management is essential for cryptographic security.
- Poor key management may easily lead to compromise of systems where the security is based on cryptography.

Key Usage
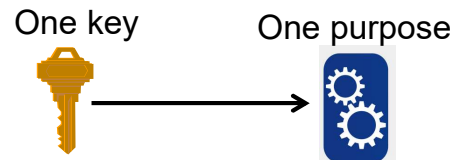
One key → One purpose

- A single key should be used for **only one purpose**
  - e.g., encryption, authentication, key wrapping, random number generation, or digital signature generation
- Using the same key for two different purposes may weaken the security of one or both purposes.
- Limiting the use of a key limits the damage that could be done if the key is compromised.
- Some uses of keys interfere with each other
  - e.g. an asymmetric key pair should only be used for either encryption or digital signatures, not both.

Types of Cryptographic Keys
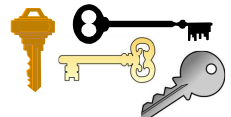
- Crypto keys are classified according to:
  - Whether they're public, private or symmetric
  - Their intended use
  - For asymmetric keys, also whether they're static (long life) or ephemeral (short life)
- 19 different types of cryptographic keys defined in: NIST Special Publication 800-57, Part 1, "Recommendation for Key Management"
 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

Crypto Period

Cryptoperiod ◄────────►

- The crypto period is the time span during which a specific key is authorized for use
- The crypto period is important because it:
  - Limits the amount of information, protected by a given key, that is available for cryptanalysis.
  - Limits the amount of exposure and damage, should a single key be compromised.
  - Limits the use of a particular algorithm to its estimated effective lifetime.
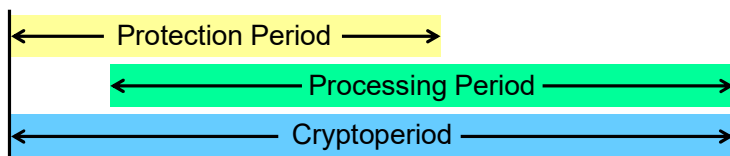

Crypto Periods

- A key can be used for <u>protection</u> and/or <u>processing</u>.
- Protection:
  - Key is used to encrypt (asymmetric or symmetric)
  - Key is used to generate a digital signature (asymmetric)
- Processing:
  - Key is used to decrypt (asymmetric or symmetric)
  - Key is used to verify a digital signature (asymmetric)
- Time frame in which key usage is allowed/recommended:
  - Protection Period (Originator-Usage Period)
  - Processing Period (Recipient-Usage Period)

◄──── Protection Period ────►
◄──────── Processing Period ────────►
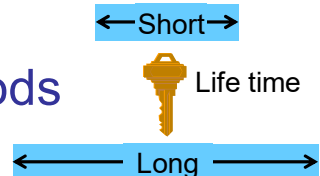◄──────── Cryptoperiod ────────►
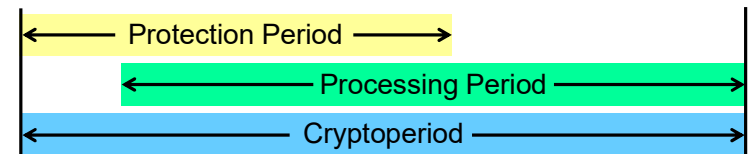
Crypto Periods

- The processing period can continue after the protection period.
- The **crypto-period** lasts from the beginning of the protection period to the end of the processing period.

◄──── Protection Period ────►
◄──────── Processing Period ────────►
◄──────── Cryptoperiod ────────►

◄─Short─►

Life time

Factors Affecting Crypto-Periods

◄──── Long ────►

- In general, as the sensitivity of the information or the criticality of the processes increases, the crypto-period should decrease in order to limit the damage resulting from compromise.
- Short crypto-periods may be counter-productive, particularly where denial of service is the paramount concern, and there is a significant overhead and potential for error in the re-keying, key update or key derivation process.
- The crypto-period is therefore a **trade-off**

Security-strength time frame (ignoring QC)
Ref: NIST SP 800-57

Security Strength		Through 2030	2031 and Beyond
< 112	Applying	Disallowed	
	Processing	Legacy-use	
112	Applying	Acceptable	Disallowed
	Processing		Legacy use
128	Applying/Processing	Acceptable	Acceptable
192		Acceptable	Acceptable
256		Acceptable	Acceptable

Key strength comparison (ignoring QC)
Ref: NIST SP 800-57

Security Strength	Symmetric key algorithms	Finite Field Cryptography FFC (e.g., DSA, D-H)	Integer Factorization Cryptography IFC (e.g., RSA)	Elliptic Curve Cryptography ECC (e.g., ECDSA)
≤ 80	2TDEA[21]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Quantum Computing's Impact on Crypto

- NIST (US National Institute of Standards and Technology) assumes powerful quantum computers can come in the 2020s
- Impact on public-key crypto:
  - RSA
  - Elliptic Curve Cryptography (ECDSA)
  - Finite Field Cryptography (DSA)
  - Diffie-Hellman key exchange
- Impact on symmetric key crypto:
  - AES ➢ Needs double key size (256 bit is OK)
- Impact on hash functions:
- SHA-2 and SHA-3 ➢ Needs double hash size (512 bit is OK)

Key Generation

- Most sensitive of all cryptographic functions.
- Prevent unauthorized disclosure, insertion, and deletion of keys.
- Automated devices that generate keys and initialisation vectors (IVs) should be physically protected to prevent:
  - disclosure, modification, and replacement of keys,
  - modification or replacement of IVs.
- Keys must be randomly chosen from the full range of the key space
  - e.g. 128 bit keys give a key space of 2^{128} different keys
  - Every key should be equally likely

When keys are not random

- Revealed by Edward Snowden 2013, NSA bribed RSA Security Inc. (prominent security company) US$ 10 Million to implement a flawed random-number generator in their BSAFE security products.
- NSA were able to predict the random numbers and regenerate the same secret keys as those used by RSA's customers.
- With the secret keys, NSA could read all data encrypted with RSA's BSAFE security product.

Schneier on Security

Blog | Newsletter | Books | Essays | News | Talks | Academic | About Me

Blog >

Random Number Bug in Debian Linux

This is a big deal.

On May 13th, 2008 the Debian project announced that Luciano Bello found an interesting vulnerability in the OpenSSL package they were distributing. The bug in question was caused by the removal of the following line of code from md_rand.c

```
MD_Update(&m,buf,j);
[ .. ]
MD_Update(&m,buf,j); /* purify complains */
```

These lines were removed because they caused the Valgrind and Purify tools to produce warnings about the use of uninitialized data in any code that was linked to OpenSSL. You can see one such report to the OpenSSL team here. Removing this code has the side effect of crippling the seeding process for the OpenSSL PRNG. Instead of mixing in random data for the initial seed, the only "random" value that was used was the current process ID. On the Linux platform, the default maximum process ID is 32,768, resulting in a very small number of seed values being used for all PRNG operations.

More info, from Debian, here. And from the hacker community here. Seems that the bug was introduced in September 2006.

More analysis here. And a cartoon.

Random numbers are used everywhere in cryptography, for both short- and long-term security. And, as we've seen here, security flaws in random number generators are really easy to accidently create and really hard to discover after the fact. Back when the NSA was routinely weakening commercial cryptography, their favorite technique was reducing the entropy of the random number generator.

Search
Powered by DuckDuckGo [Go]
○ blog ○ essays ○ whole site

Subscribe

About Bruce Schneier

I've been writing about security issues on my blog since 2004, and in my monthly newsletter since 1998. I write books, articles, and academic papers. Currently, I'm the Chief Technology Officer of IBM Resilient, a fellow at Harvard's Berkman Center, and a board member of EFF.

Compromise of keys and keying material

- Key compromise occurs when it is known or suspected that an unauthorized entity has obtained a secret/private key.
- When a key is compromised, immediately stop using the secret/public key for protection, and revoke the compromised key (pair).
- A compromised key **might** be used for continued processing of protected information.
  - In this case, the entity that uses the information must be made fully aware of the risks involved.
  - Continued key usage for processing depends on the risks, and on the organization's Key Management Policy.
- The worst form of key compromise is when a key is compromised without detection.

Key States, Transitions and Phases
Ref: NIST SP 800-57
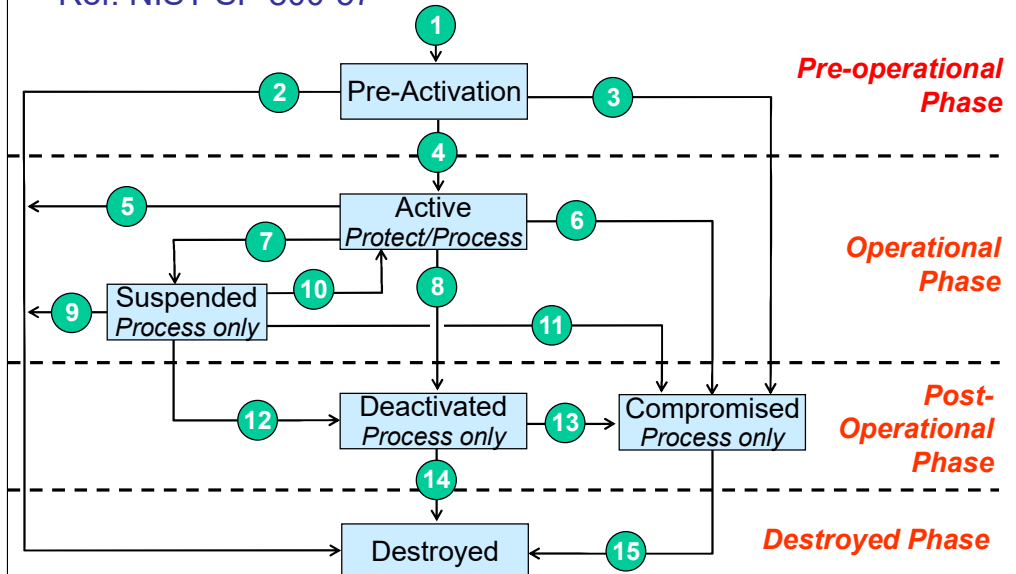
Key Protection

- Active **keys should be**
  - accessible for authorised users,
  - protected from unauthorised users
- Deactivated keys must be kept as long as there exists data protected by keys. Policy must specify:
  - Where keys shall be kept
  - How keys shall be kept securely
  - How to access keys when required

Key Protection Examples

- Symmetric ciphers
  - Never stored or transmitted 'in the clear'
  - May use hierarchy: session keys encrypted with master key
  - Master key protection:
    - Locks and guards
    - Tamper proof devices
    - Passwords/passphrases
    - Biometrics
- Asymmetric ciphers
  - Private keys need confidentiality protection
    (see above "Master Key")
  - Public keys need integrity/authenticity protection
    (see next section "PKI")

Key destruction

- No key material should reside in volatile memory or on permanent storage media after destruction
- Key destruction methods, e.g.
  - Simple delete operation on computer
    - may leave undeleted key e.g. in recycle bin or on disk sectors
  - Special delete operation on computer
    - that leaves no residual data, e.g. by overwriting
  - Magnetic media degaussing
  - Destruction of physical device e.g. high temperature
  - Master key destruction which logically destructs subordinate keys
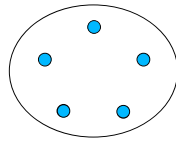
PKI   Public-Key Infrastructure

Cryptography solves security problems in open networks, … but creates key distribution challenges.


Public-key cryptography simplifies the key distribution, … but requires a PKI which creates trust management challenges.

Key distribution: The challenge

- Network domain (e.g. the Internet) with n nodes

- Every pair of nodes must be able to communicate securely under cryptographic protection
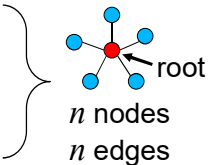- How many secure key **distributions** are needed ?
  - Symmetric secret keys: **Confidentiality** required.

    $n(n-1)/2$ distributions, quadratic growth.
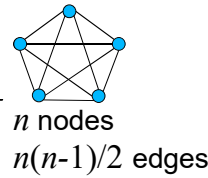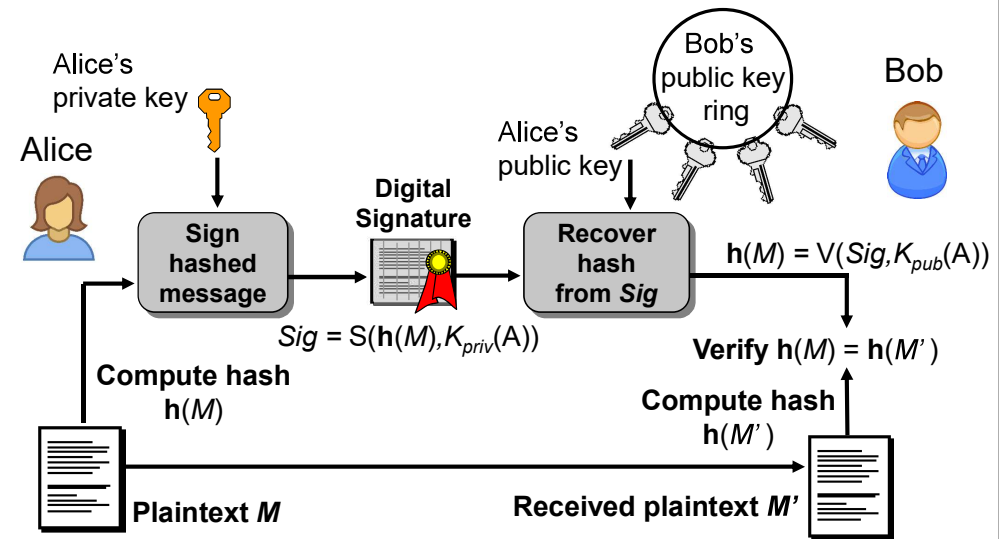
    Impractical in open networks.
  - Asymmetric public keys: **Authenticity** required.

    $n(n-1)/2$ distributions of pub. keys, quadratic growth

    Impractical in open networks

    n nodes
    $n(n-1)/2$ edges

  - Asymmetric public keys with PKI: **Authenticity** required.
    - 1 root public key distributed to n parties
    - linear growth
   - … easier, but still relatively challenging

    root

    n nodes
    n edges

Recapitulation: Digital signature

$Sig = S(\mathbf{h}(M), K_{priv}(A))$

Compute hash $\mathbf{h}(M)$

Plaintext M

$\mathbf{h}(M) = V(Sig, K_{pub}(A))$

Verify $\mathbf{h}(M) = \mathbf{h}(M')$

**Compute hash** $\mathbf{h}(M')$

Received plaintext M'

Public-key rings and registries don't work

- Impractical for an entity (individual or system) to keep a public-key ring with the public keys of all other entities they might want to communicate with.
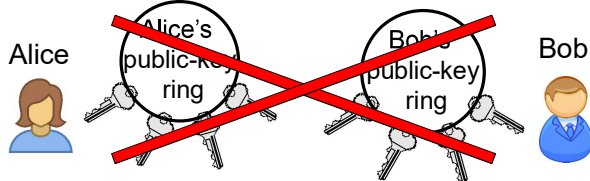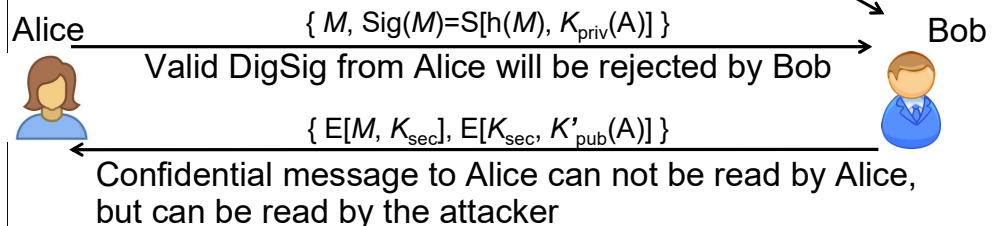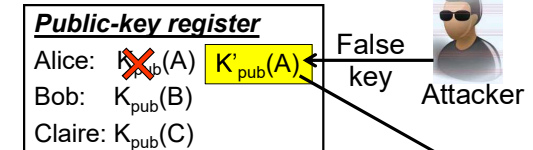

- Also impractical to have a register on the Internet with everyone's public keys

Public-key register
Alice: $K_{pub}(A)$
Bob: $K_{pub}(B)$
Claire: $K_{pub}(C)$
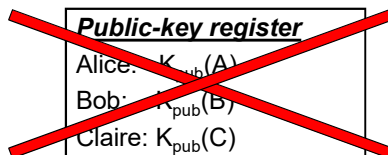
- There must be a better way: PKI !

Problem of non-authentic public keys

- Assume that public keys are stored in a public register
- What is the consequence if attacker replaces Alice's public key the register?

Public-key register
Alice: $K_{pub}(A)$ $K'_{pub}(A)$ ← False key ← Attacker
Bob: $K_{pub}(B)$
Claire: $K_{pub}(C)$

Alice

$\{ M, Sig(M)=S[\mathbf{h}(M), K_{priv}(A)] \}$
Valid DigSig from Alice will be rejected by Bob

Bob

$\{ E[M, K_{sec}], E[K_{sec}, K'_{pub}(A)] \}$
Confidential message to Alice can not be read by Alice, but can be read by the attacker

- Broken public-key authenticity breaks security assumptions

PKI Public-Key Infrastructure

- Due to the spoofing problem, public keys must be digitally signed and distributed as "public-key certificates".

- The main purpose of a PKI is to ensure authenticity of public keys and simplify the key distribution.

- PKI consists of:
  - **Policies** (to define the rules for managing certificates)
  - **Technologies** (to implement the policies and generate, store and manage certificates)
  - **Procedures** (related to key management)
  - **Trust model of public-key certificates** (how the certificates are cryptographically linked to each other)

Public-Key Certificates

- A public-key certificate is a record of data, including the subject distinguished name and its public key, all digitally signed by a CA (Certificate Authority).
- Binds name to public key
- An authentic copy of the CA's public key is needed in order to validate the certificate.
- **Relying party** validates the certificate (i.e. verifies that the digital signature is correct).

X.509 Digital Certificate
- Version
- Serial Number
- Algorithm Identifier
- Issuer CA
  - Distinguished Name
- Subject
  - **Distinguished Name**
  - **Public Key**
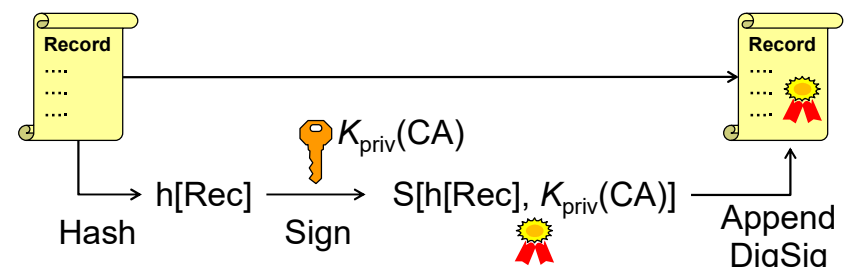- Validity Period
- Extensions

Digital Signature

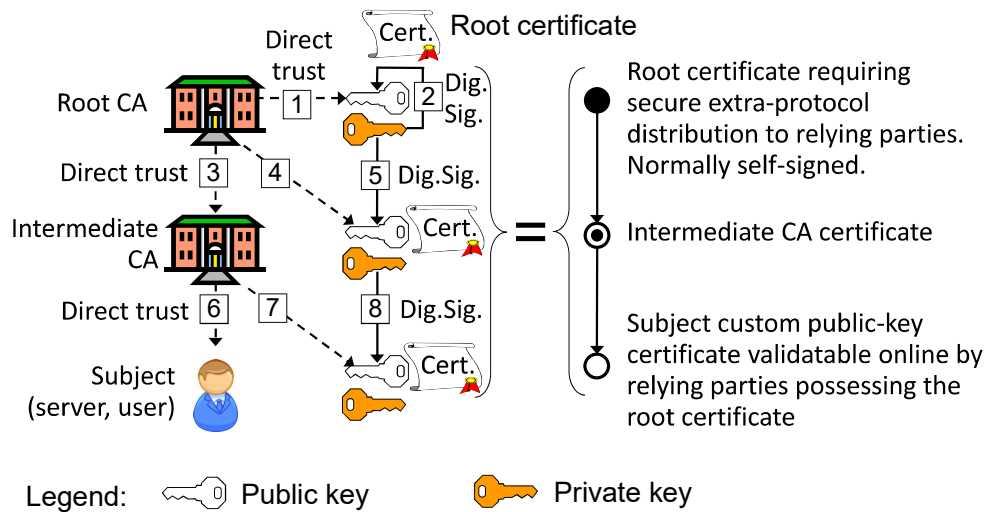# Example of X.509 certificate
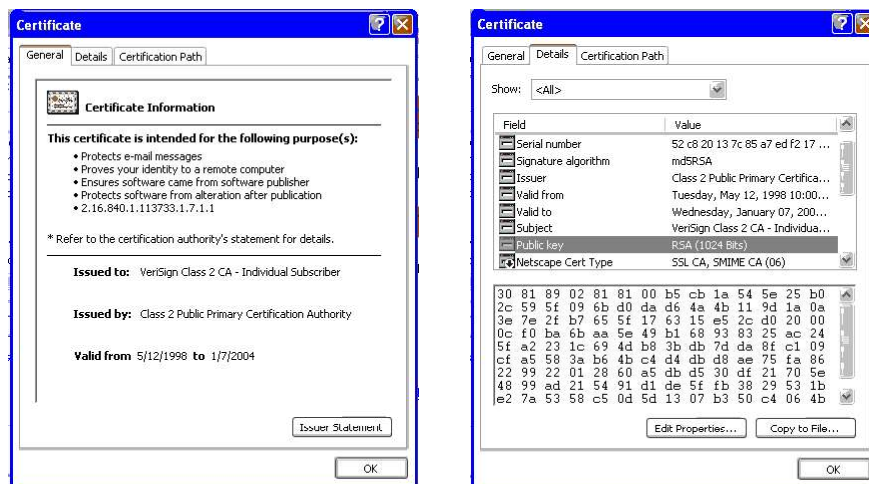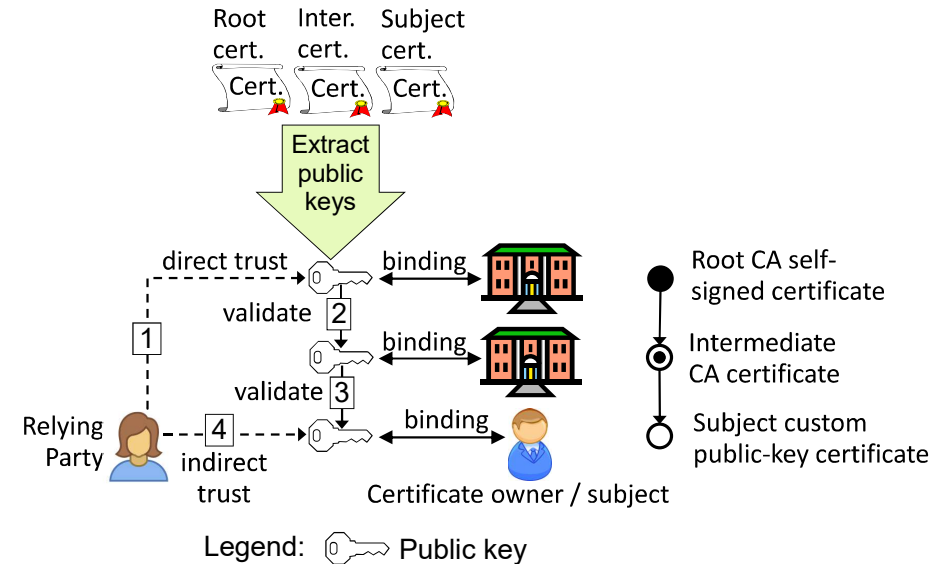
How to generate a digital certificate?

1. Assemble all the date (including entity's subject name and its public key) in a single record Rec
2. Hash the record
3. Sign the hashed record
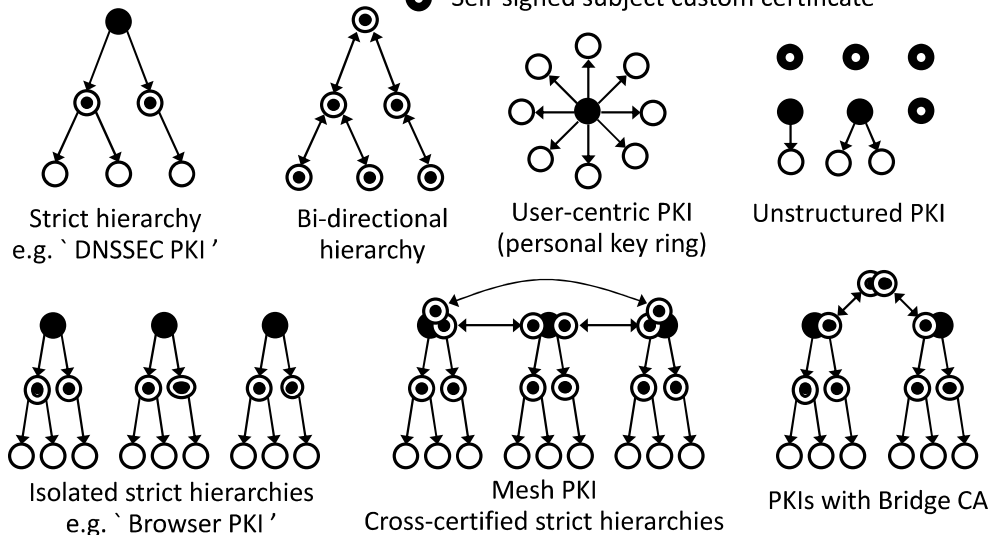4. Append the digital signature to the record

Record
....
....
....

$K_{priv}(CA)$

Record
....
....
....

$h[Rec]$ → $S[h[Rec], K_{priv}(CA)]$ → Append DigSig

Hash   Sign

PKI certificate generation

Root certificate

Root certificate requiring secure extra-protocol distribution to relying parties. Normally self-signed.

● Root certificate requiring secure extra-protocol distribution to relying parties. Normally self-signed.

◉ Intermediate CA certificate

○ Subject custom public-key certificate validatable online by relying parties possessing the root certificate

Direct trust — 1

Root CA

2 — Dig. Sig.

Direct trust 3 — 4

5 Dig.Sig.

Intermediate CA — Cert.

Direct trust 6 — 7

8 Dig.Sig.

Cert.

Subject (server, user)

Legend: ⚿ Public key   🔑 Private key

Certificate and public key validation

Root cert.   Inter. cert.   Subject cert.

Extract public keys

direct trust   binding   Root CA self-signed certificate ●

validate 2   binding   Intermediate CA certificate ◉

1

validate 3   binding   Subject custom public-key certificate ○

Relying Party   4   indirect trust

Certificate owner / subject

Legend: ⚿ Public key

PKI Trust Models

Legend:
- ● Self-signed root CA certificate
- ◉ CA-signed intermediate CA certificate
- ○ CA-signed custom (leaf) certificate (cannot sign)
- ◉ Self-signed subject custom certificate

Strict hierarchy e.g. `DNSSEC PKI '

Bi-directional hierarchy

User-centric PKI (personal key ring)

Unstructured PKI

Isolated strict hierarchies e.g. `Browser PKI '

Mesh PKI
Cross-certified strict hierarchies

PKIs with Bridge CA

Meaning of Trust for PKI

- **Trustworthy**: When it is objectively secure and reliable
- **Trusted**:   When we decide to depend on it

- A root certificate is **trustworthy** when it has been received securely from a honest + reliable CA.
- A root certificate is **trusted** when it is being used to validate other certificates.

- Ideally, only trustworthy root certificates should be trusted
- In reality, many untrustworthy certificates are trusted.
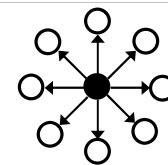
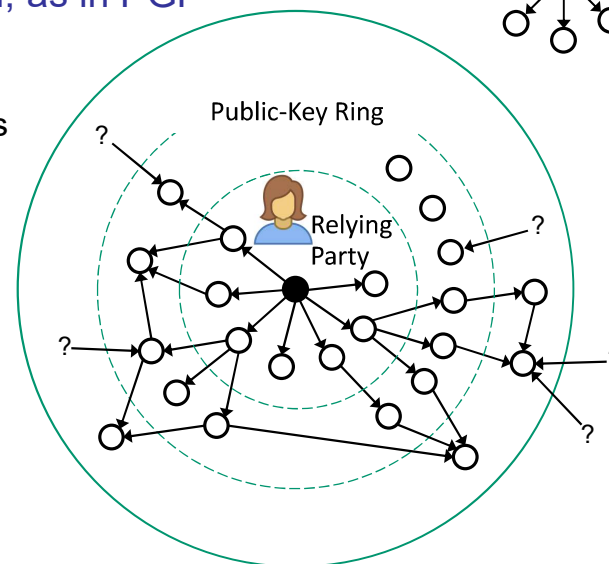# PKI trust models
Strict hierarchical model
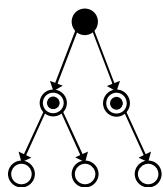
- Advantages:
  - works well in highly-structured setting such as military and government
  - unique certification path between two entities (so finding certification paths is trivial)
  - scales well to larger systems
- Disadvantages:
  - need a trusted third party (root CA)
  - 'single point-of-failure' target
  - If any node is compromised, trust impact on all entities stemming from that node
  - Does not work well for global implementation (who is root TTP?)

Web of trust PKI model
User-centric model, as in PGP

- Each party signs public keys of others whose keys have been verified to be authentic.
- Public keys signed by trusted people can be considered authentic too.

PKI trust models
User-centric model
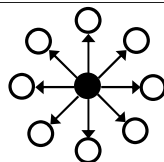
- Each user is **completely responsible** for deciding which public keys to trust
- Example: *Pretty Good Privacy (PGP)*
  - 'Web of Trust'
  - Each user may act as a CA, signing public keys that they will trust
  - Public keys can be distributed by key servers and verified by fingerprints
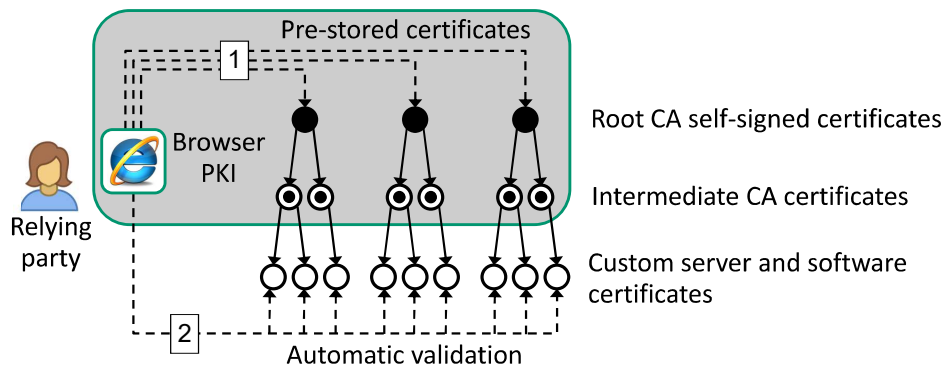 - OpenPGP Public Key Server: http://pgpkeys.mit.edu:11371/

PKI trust models
User-centric model
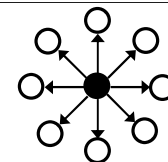
- Advantages:
  - Simple and free
  - Works well for a small number of users
  - Does not require expensive infrastructure to operate
  - User-driven grass-root operation
- Disadvantages:
  - More effort, and relies on human judgment
    - Works well with technology savvy users who are aware of the issues. Does not work well with the general public
  - Not appropriate for more sensitive and high risk areas such as finance and government

The Browser PKI
(PKI based on the X.509 certificates)

Pre-stored certificates

Root CA self-signed certificates

Intermediate CA certificates

Custom server and software certificates

Automatic validation

Relying party

Browser PKI

The browser PKI model consists of isolated strict hierarchies where the (root) CA certificates are installed as part of the web browser. New roots and trusted certificates can be imported after installation
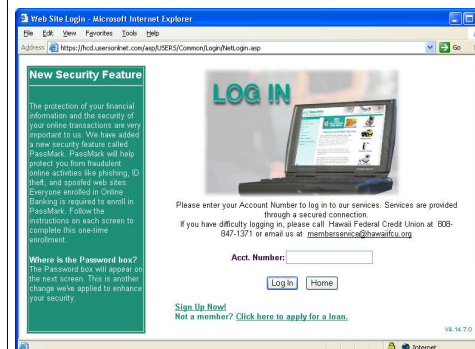
Browser PKI and malicious certificates

- The browser automatically validates certificates by checking: certificate name = domain name
- Criminals buy legitimate certificates which are automatically validated by browsers
  - Legitimate certificates can be used for malicious phishing attacks, e.g. to masquerade as a bank
  - Malicious sites can have legitimate certificates !!!
- Server certificate validation is only syntactic authentication, **not** semantic authentication
  - Users who don't know the server domain name can *a priori* not know if it's a 'good' domain

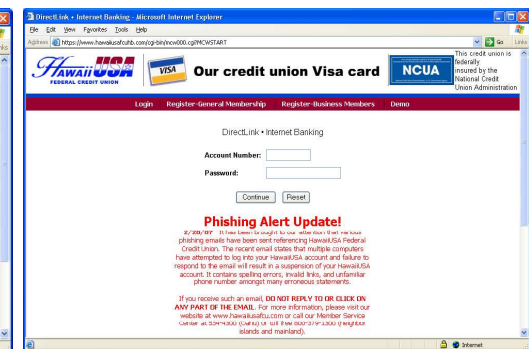# Browser PKI root certificate installation

- Distribution of root certificates should happen securely out-of-band (not online)
  - But root certificate distribution is typically done by downloading browser SW
  - Is this secure ?
- Users must in fact trust the browser and OS vendors who install the root certificates,
  - Example: *Chrome, Mozilla Firefox* and *Microsoft Edge*
  - Trust in the root CAs is only implicit
- Browser vendors decide which CA root certs to install
  - This is an important consideration for security
  - How do we know that a browser only contains trustworthy certificates ?

Phishing and fake certificates
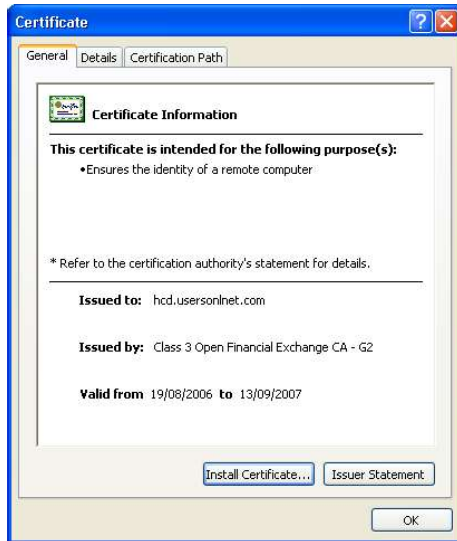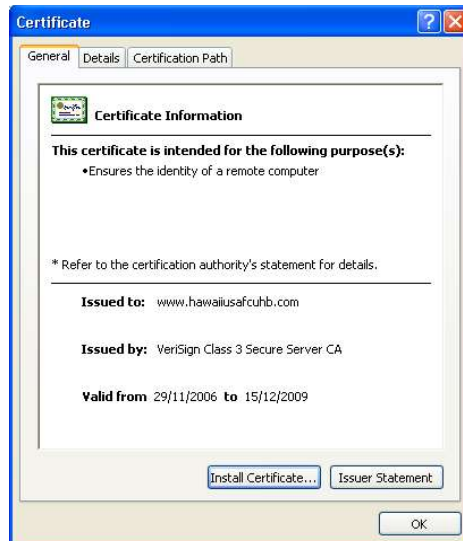# Hawaii Federal Credit Union

Genuine bank login
https://hcd.usersonlnet.com/asp/USERS/Common/Login/NettLogin.asp

Fake bank login
https://hawaiiusafcuhb.com/cgi-bin/mcw00.cgi?MCWSTART
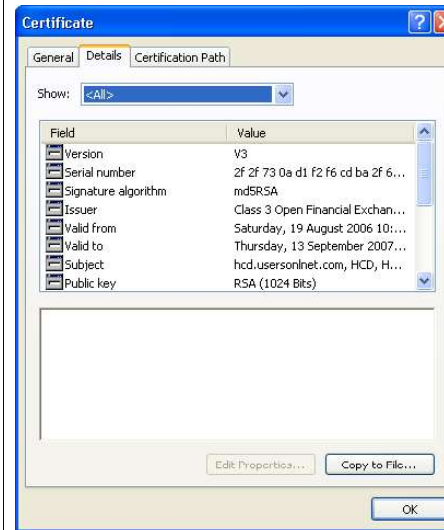
Authentic and Fake Certificates
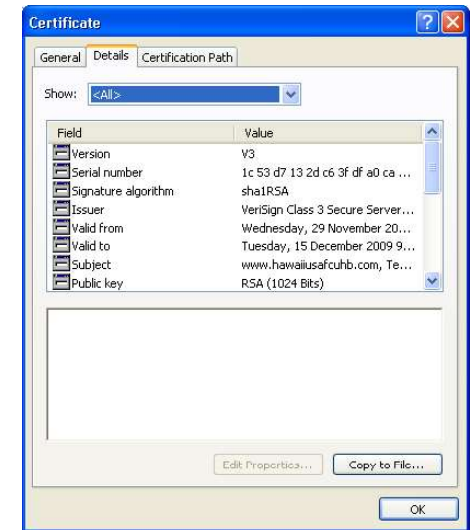

Genuine certificate   "Fake" certificate

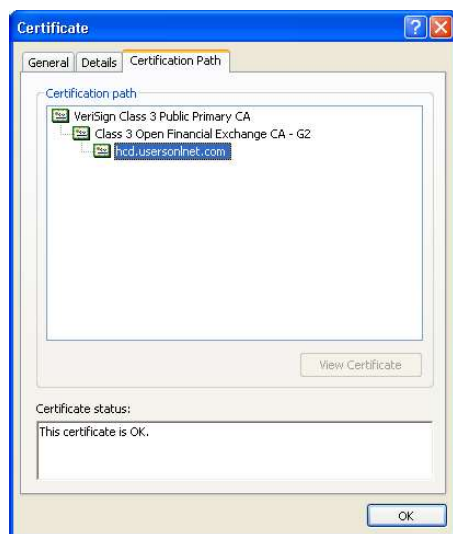# Certificate comparison 2

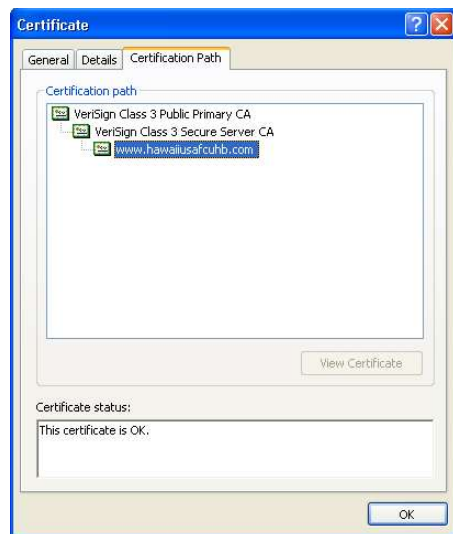Genuine certificate   "Fake" certificate
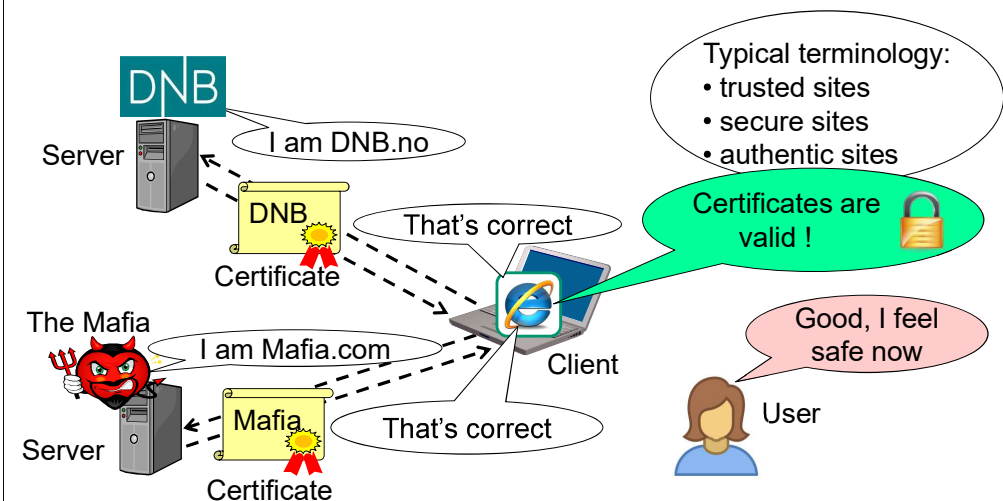
Certificate comparison 3

Genuine certificate   "Fake" certificate

Meaningless Server Authentication
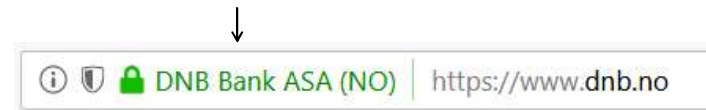
Extended validation certificates

- Problem with simple certificates:
  - Can be requested by anonymous entities (nowadays even free of charge)
- EV (Extended Validation) certificates require registration of legal name of certificate owner.
- Provides increased assurance in website identity.
- However, EV certificates are only about identity, not about honesty, reliability or anything normally associate with trust.
- Even the Mafia can buy EV certificates through legal businesses that they own.

Extended validation certificates
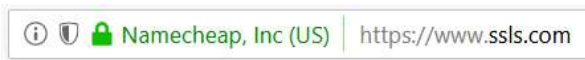
a) "Standard" certificate

Legal name of website owner is displayed on the address bar when using Extended Validation Certificates.

↓

b) Extended validation certifiate
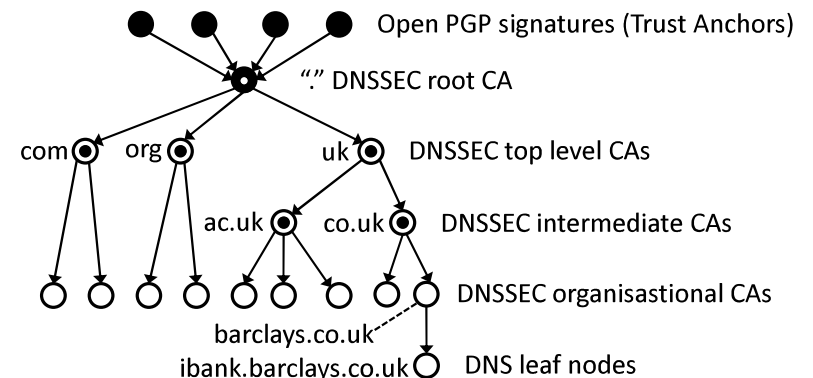
Problems of EV Certificates

- Domain name and owner name not always equal

- If an attacker is able to register a similar business (even in a different country), most users will not recognize the difference
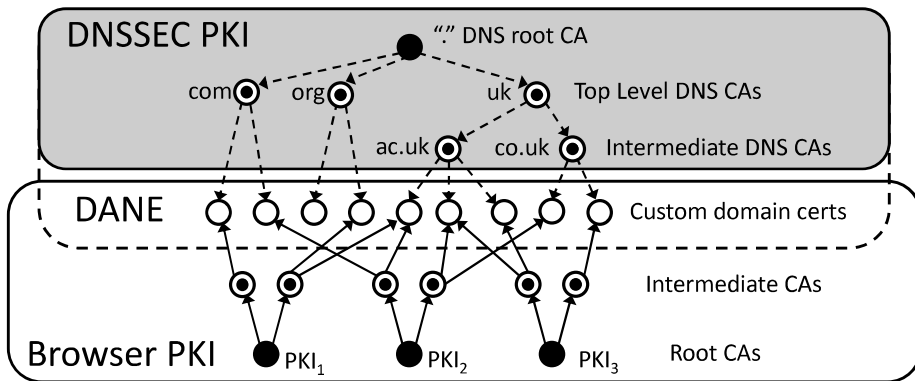
DNSSEC PKI

- The DNS (Domain Name System) is vulnerable to e.g. cache poisoning attacks resulting in wrong IP addresses being returned.
- DNSSEC designed to provide digital signature on every DNS reply
- Based on PKI with a single root.

DNSSEC PKI vs. Browser PKI

- CAs in the Browser PKI can issue certificates for arbitrary domains
- DANE: DNSSEC-based Authentication of Named Entities
  - Certificates for custom domains issued under DNSSEC PKI
  - Alternative to B-PKI, standards exist, but not widely deployed

Security fails when CAs are compromised

- CA DigiNotar hacked in 2011
- Many fake certificates (e.g. *.google.com) were created by the hackers
- If a single CA is compromised, any company on the Internet can be spoofed.
- A new method called CAA (CA Authorization) introduced in 2018 prevents a single CA compromise from being a threat to the whole Internet.

KIM ZETTER SECURITY 09.20.11 03:05 PM

DIGINOTAR FILES FOR BANKRUPTCY IN WAKE OF DEVASTATING HACK

DigiNotar®
A GMO VASCO COMPANY

Go to …

A Dutch certificate authority that suffered a major hack attack this summer has been unable to recover from the blow and filed for bankruptcy this week.

PKI Summary

- Public key cryptography needs a PKI in order to be practical

- It is complex and expensive to operate a robust PKI

- PKI services are called 'Trust Services' in EU's Digital Agenda
  - Intended as a security foundation for e-Id and e-Services in the EU

- Establishing initial trust in PKIs has a cost, because it is expensive to use secure out-of-band channels needed for distributing root certificates

- The Browser PKI is the most widely deployed PKI thanks to the distribution of root certificates with web browsers

- Security of PKI depends on the integrity of the CAs

End of lecture