CS5285
Information Security for eCommerce

Lecture 7

Dr. Gerhard Hancke

CS Department

City University of Hong Kong

Reminder of previous lecture

- Authentication
  - What is needed for entity authentication?
  - Data origin authentication (sure who generated)
    - MAC/Encryption/Signature
    - Stops masquerade attacks
  - Freshness (is the message generated now?)
    - Nonce (Random or counter/sequence)/Timestamp
    - Stops replay
  - We looked at protocol examples…
    - Also remember the reflection attack

Today's Lecture

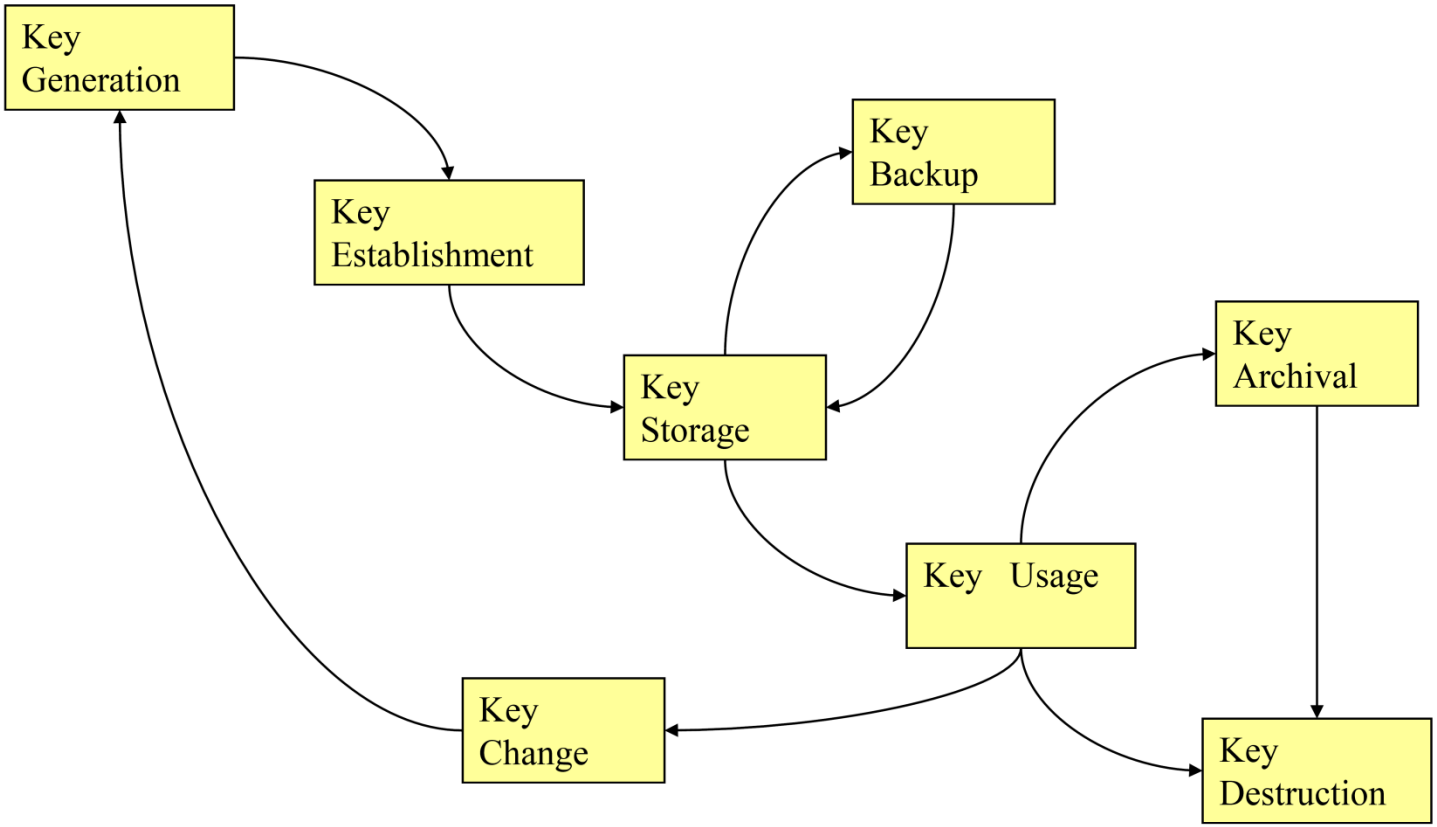
- Key Management
  - For all crypto we need keys (most important)
  - Symmetric key management
  - Asymmetric key management (Certificates)
- CILO1,CILO2, CILO3 CILO4

  (identify requirements, core threat, design and evaluation)

Credit to Keith Martin RHUL (borrowing few slides from his lecture notes – 6, 45-47)

Key Management

What is Key Management?



Hardware Security Module (HSM)

- HSM is trusted part of system
- Key generation and storage
- Features:
  - Asymmetric Crypto (Signing/Encryption)
  - Symmetric Crypto (Encryption/MAC)
  - Hashes/KDF
  - Random numbers (True random, DRBG)
- Secure networking (TLS), Tamper resistant

Symmetric-key protocols

- The use of symmetric-key cryptography to produce a shared symmetric secret key.
- The protocols can be classified as:
  - Directly communicating entities
  - Use of a Key Distribution Centre (KDC)
  - Use of a Key Translation Centre (KTC)

Directly communicating entities

- The case where two entities directly communicate to establish keys.
- Must take place using a secure channel (e.g. using an existing shared secret key or mutually trusted copies of each others' public keys).

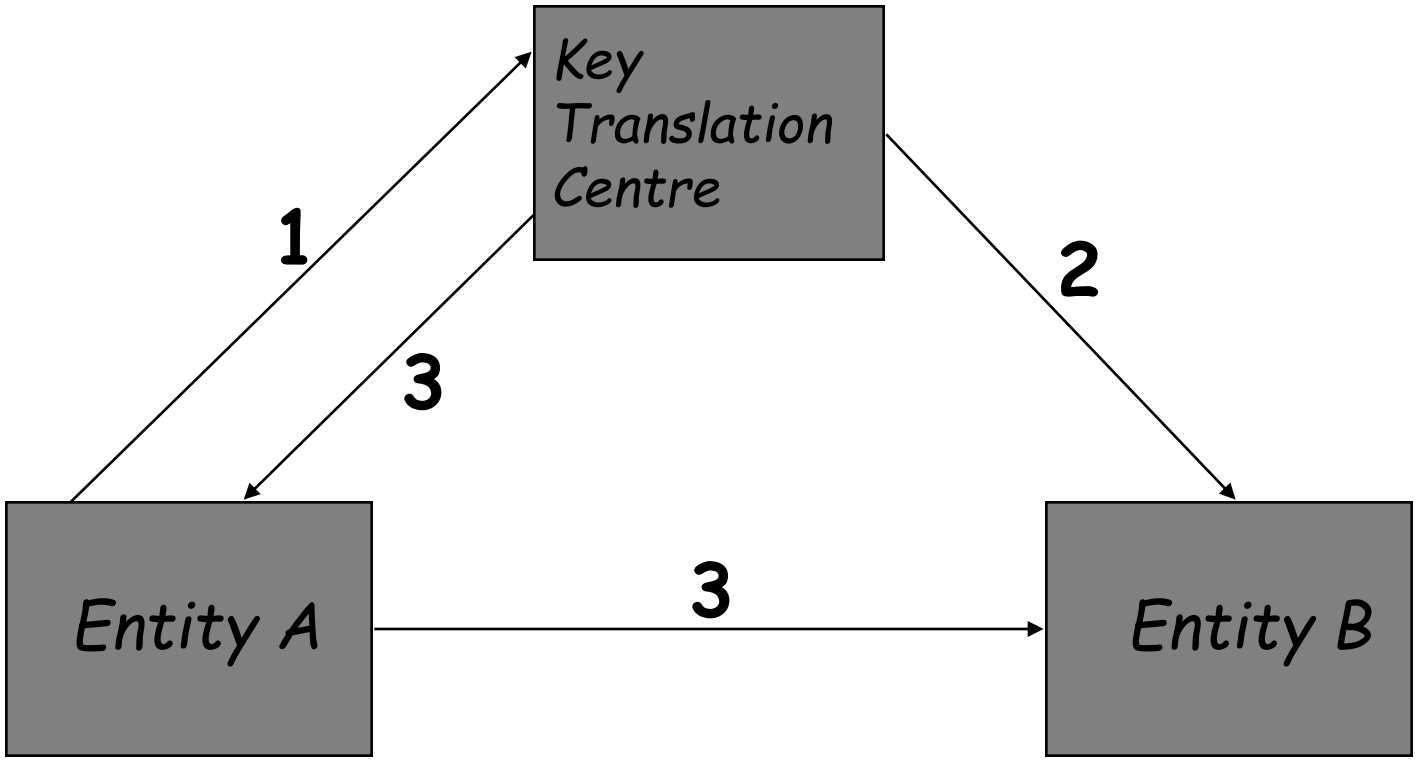
Distribution within a domain

- Two possible cases:
  - asymmetric techniques used, or
  - symmetric techniques used.
- In first case certificates may need to be distributed. Entities either contact their authority for certificates, or two entities may exchange them directly.
- In second case - use a KDC or a KTC.

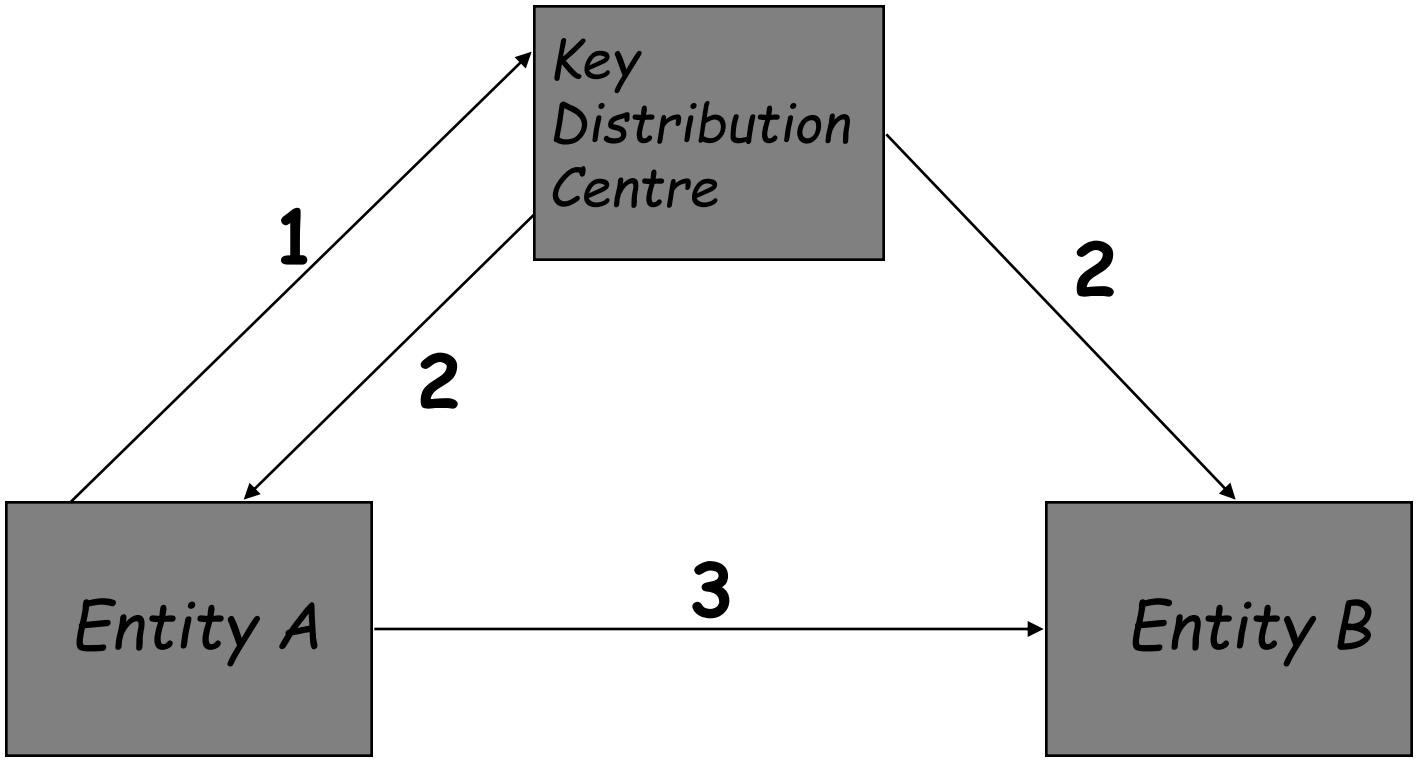
Distribution within a domain

- *Key Translation Centre* (KTC): An entity trusted to transport keys between entities that share keys with the KTC.
  - Example of a key transport service.
- *Key Distribution Centre* (KDC): An entity trusted to generate and distribute keys to entities that share keys with the KDC.
  - Arguably example of a key agreement service.

Key Translation Centre



Key Distribution Centre



Terms for key management

- *Key establishment*: Process of making a secret key available to multiple entities.
- *Key control*: Ability to a choose a key's value.
- *Key agreement*: Process of establishing a key in such a way that neither entity has key control.
- *Key transport*: Process of securely transferring a key from one entity to another.
- *Key confirmation*: Assurance that another entity has a particular key.

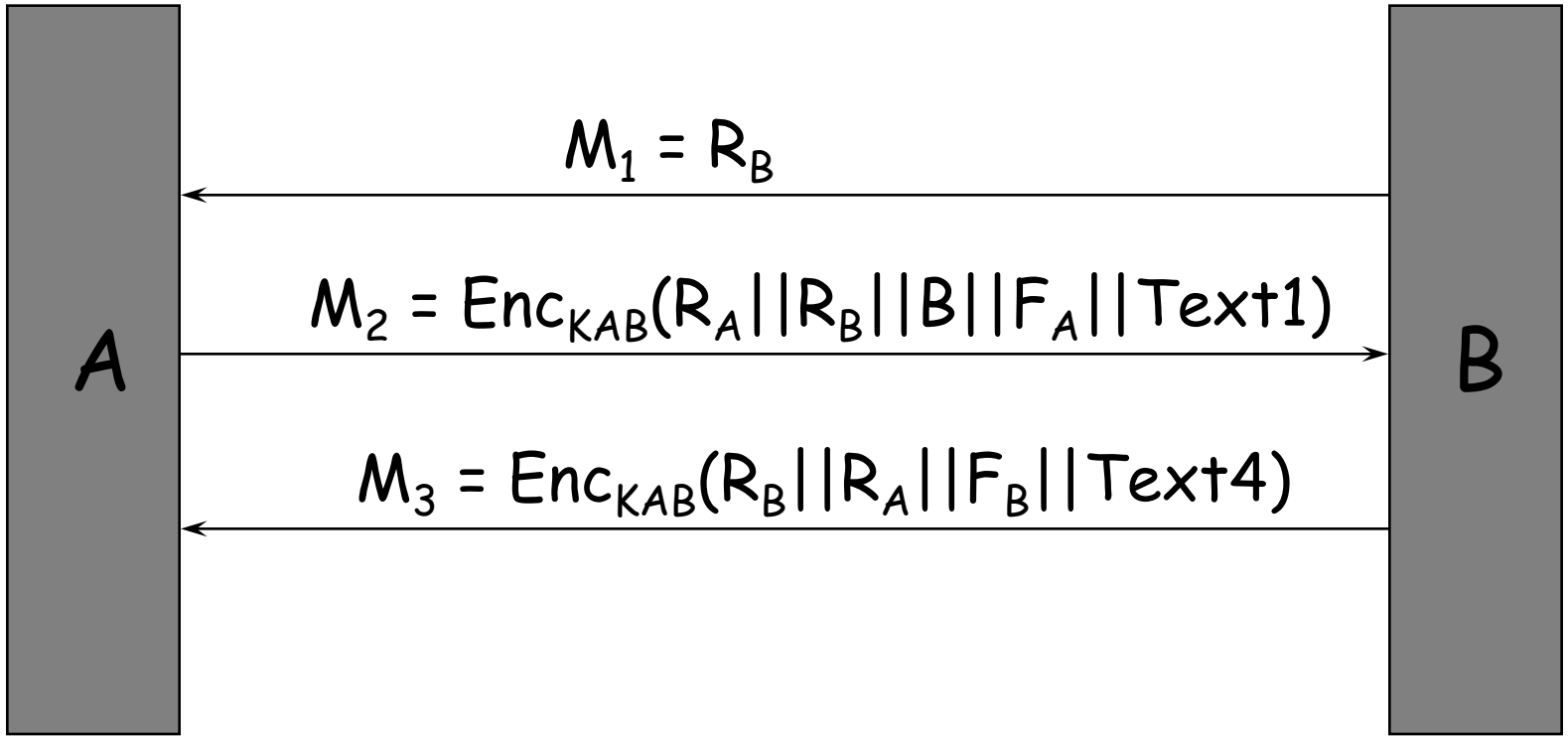
Who has the key at the end?

- We can set two requirements of key establishment
  - *Implicit key authentication to A*: The assurance for *A* that only another identified entity can possess a key. This is the basic security requirement.
  - *Explicit key authentication to A*: The assurance for *A* that only another identified entity possesses a key. This is a more stringent security requirement.

Key establishment mech. 6

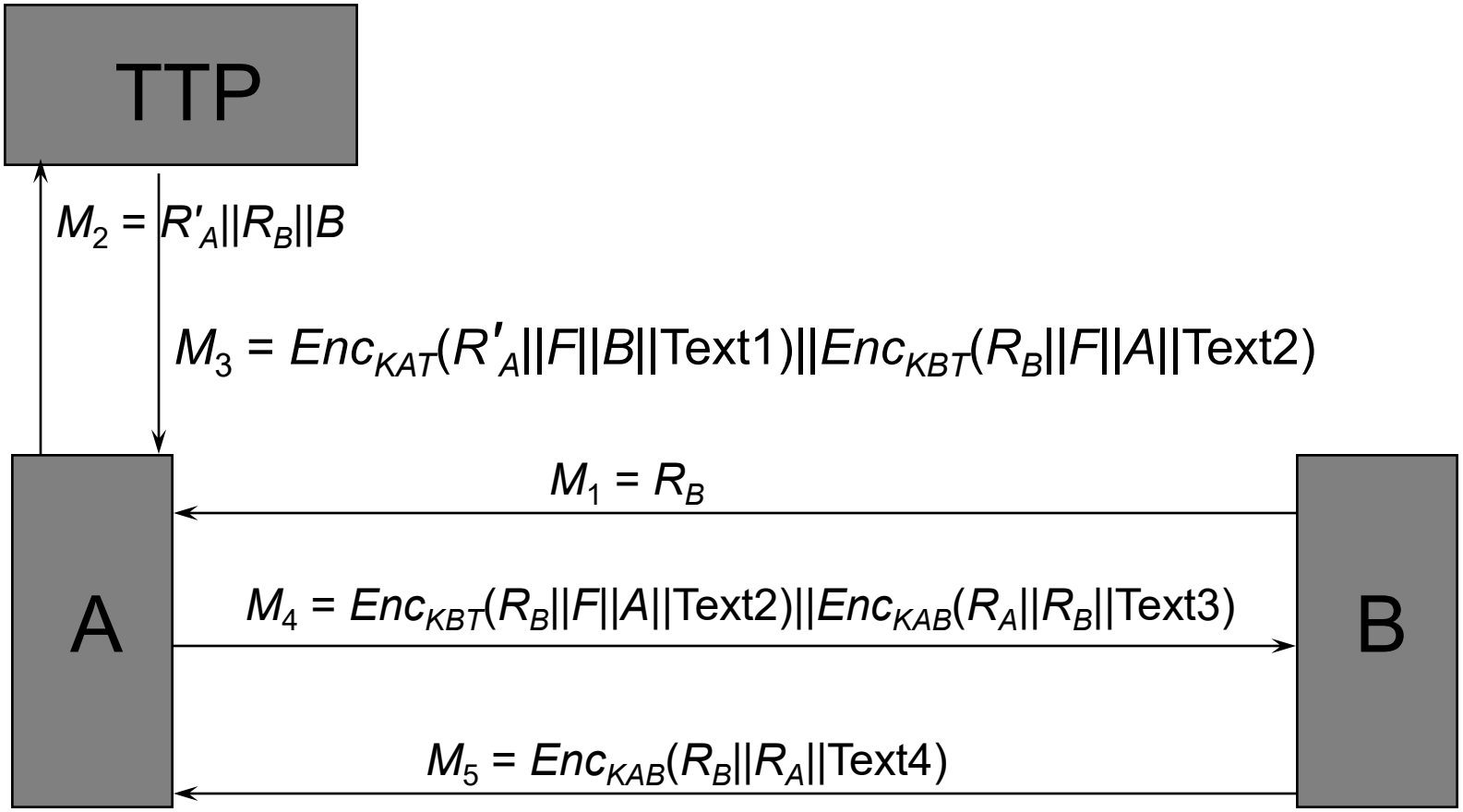
- Directly communicating entities.
- *A*, *B* must share a secret key *KAB*.
- R_A and R_B are nonces.
- F_A and F_B contain keying material.

Key establishment mech. 6

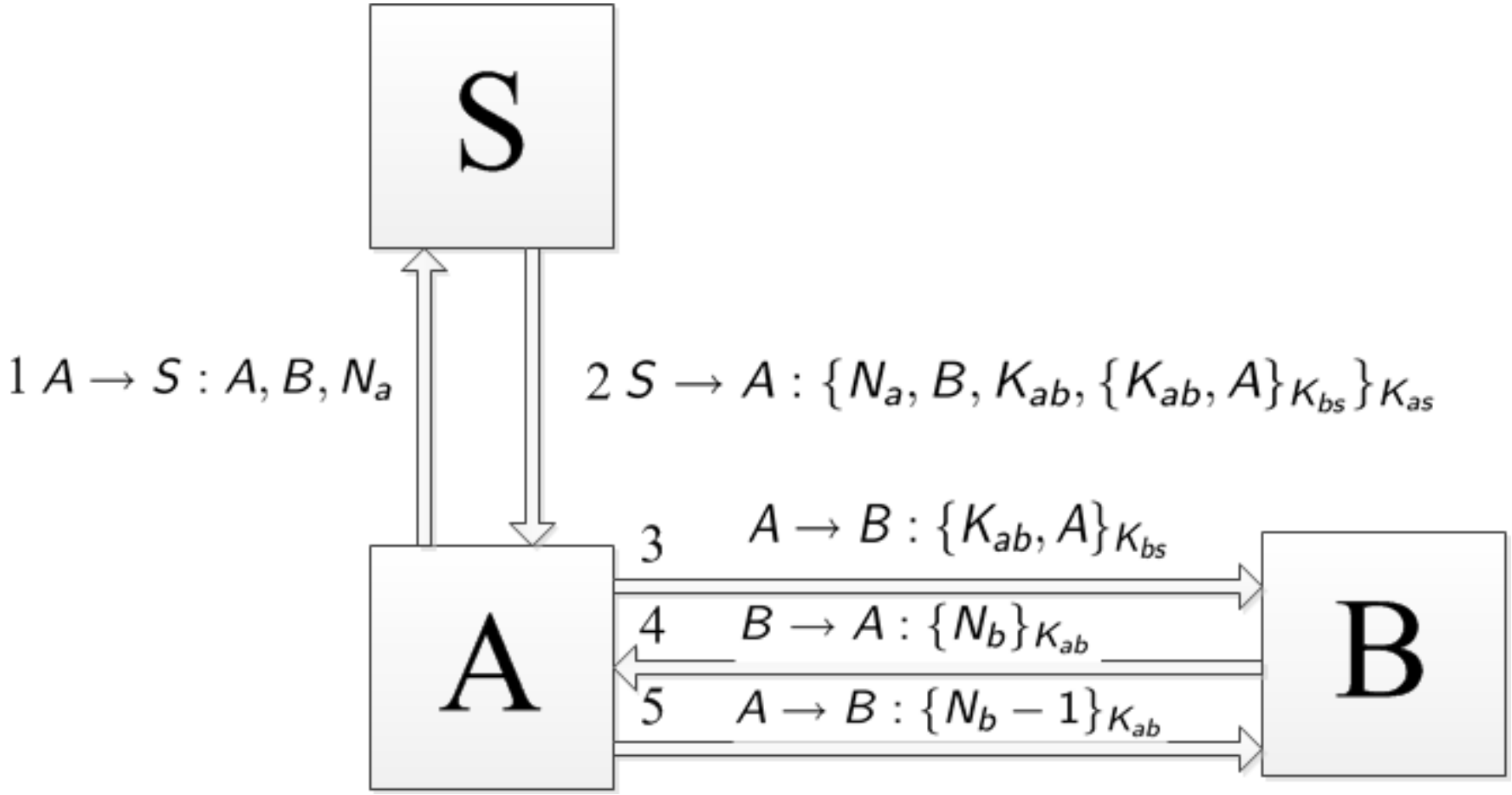


- Key K typically a hash of F_A and F_B.

Key establishment mech. 9



Remember from last lecture….



Public-key protocols

- The use of public-key cryptography to produce a shared symmetric secret key.
- The protocols can be classified as:
  - Key transport protocols (typically involving public-key encryption and digital signatures)
  - Key agreement protocols (indirectly specified and mostly based on the Diffie-Hellman protocol)

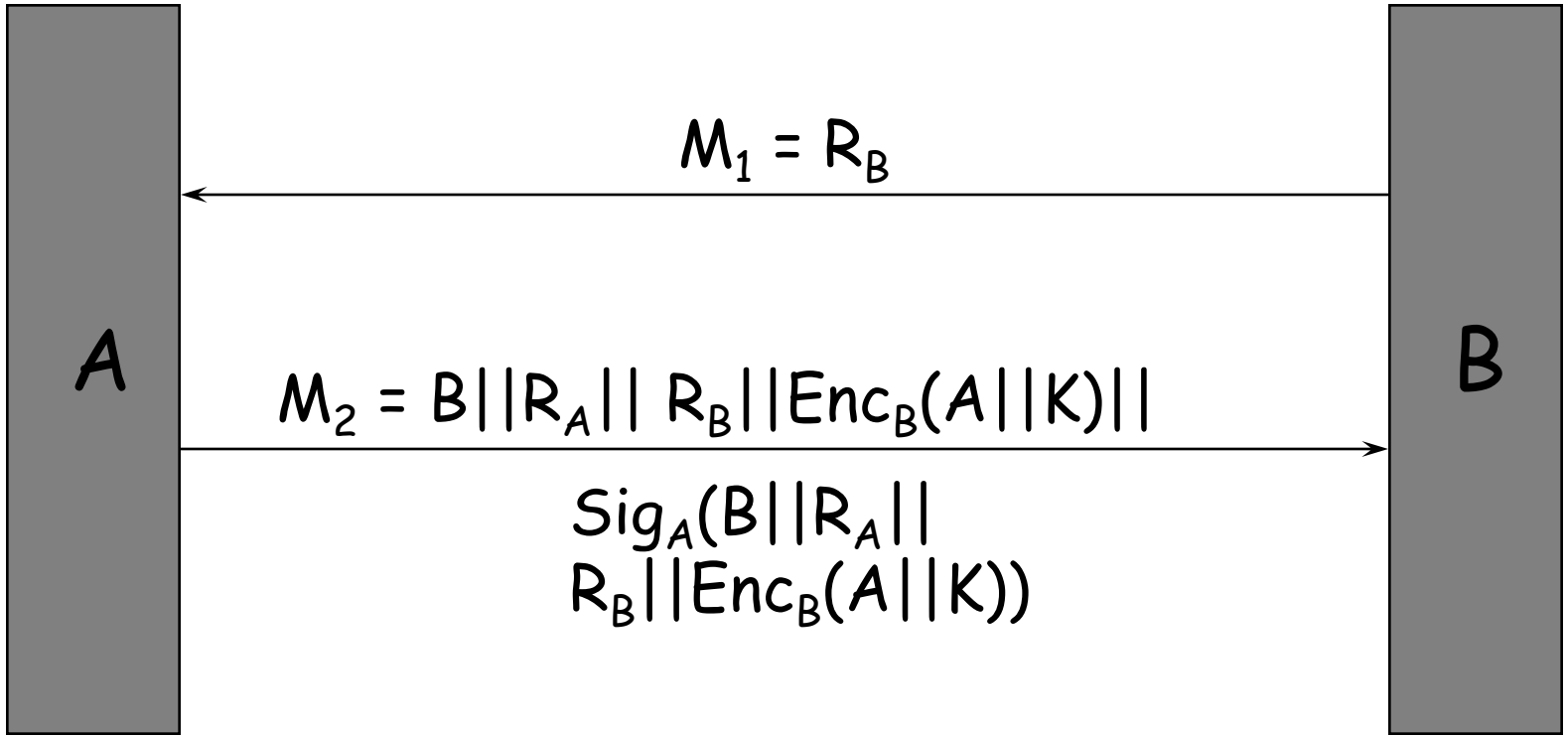
Notation

- *A* and *B* are (identifiers for) two entities who wish to engage in an authentication protocol.
- R_A is a random nonce generated by *A*.
- $Enc_A(X)$ denotes the encryption computed with *A*'s public key on the data *X*.
- $Sig_A(X)$ denotes the signature (with appendix) computed by *A* on the data *X*.

Key transport mech. 4

- Key transport protocol.
- *A*, *B* must have authenticated copies of each others public keys.
- R_A and R_B are nonces.
- The notion of keying material has been replaced with simply a key *K*.

Key transport mech. 4



Last comment: Key hierarchies

- Key are often organised in a hierarchy. Keys in one level used to protect or generate keys in next layer down.
- Only lowest layer keys (*session keys*) used for data security.
- Top layer key is *master key*. Must be protected with care.

Why use a key hierarchy

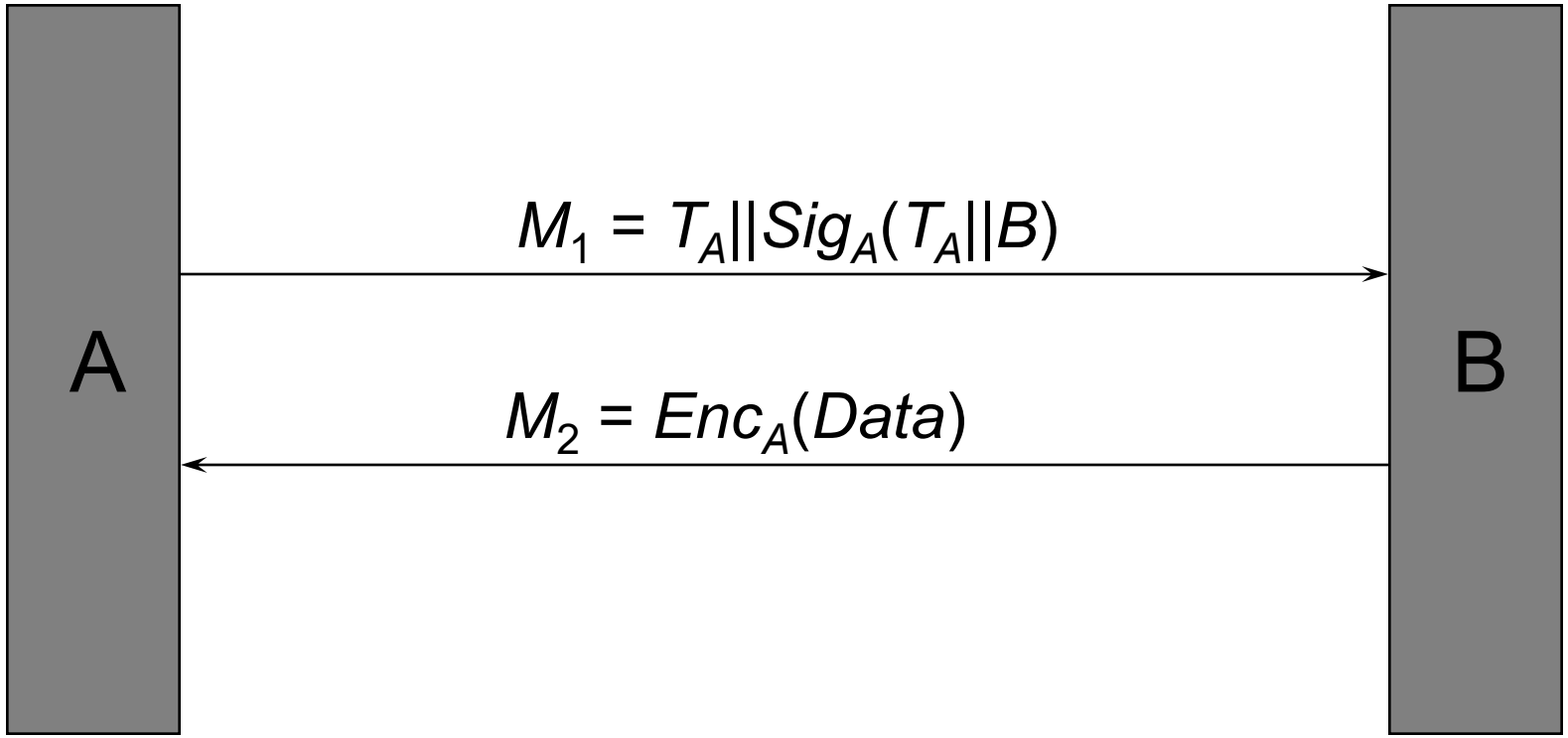
- The more we use a key the more likely it is to be compromised….
- Key establishment from nothing is hard…
- Use top layer keys sparingly (long lifetime)
  - These keys used for only for key establishment
- Low level keys used often (short lifetime)
  - Compromise of a session key of limited significance.

Example: Simple payment card

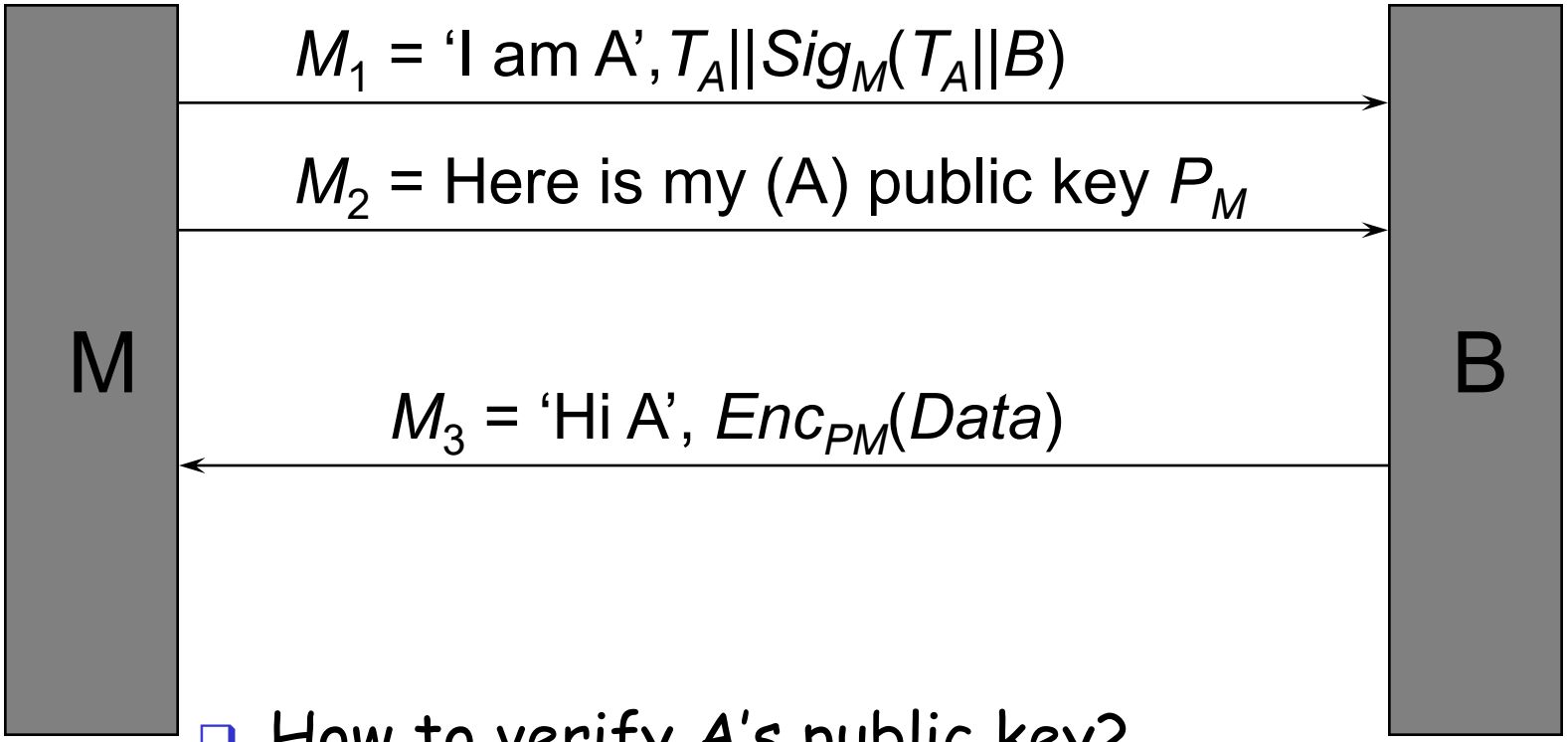
- Closed system
  - Cards/readers all controlled by same entity
- What keys do we need?
  - Card and reader need to share a key
- Problem?
  - Reader cannot store each possible card key!
  - So give all cards the same key?
  - One compromised card = completely broken system!
- The reader contains system master key
  - Card contains card key
  - Reader use master key and UID to derive card key
  - Reader and card communicate….
- If card compromised then only limited damage.

Public Key (Certificate) Management

Remember….we would like to use signatures (and PKE)



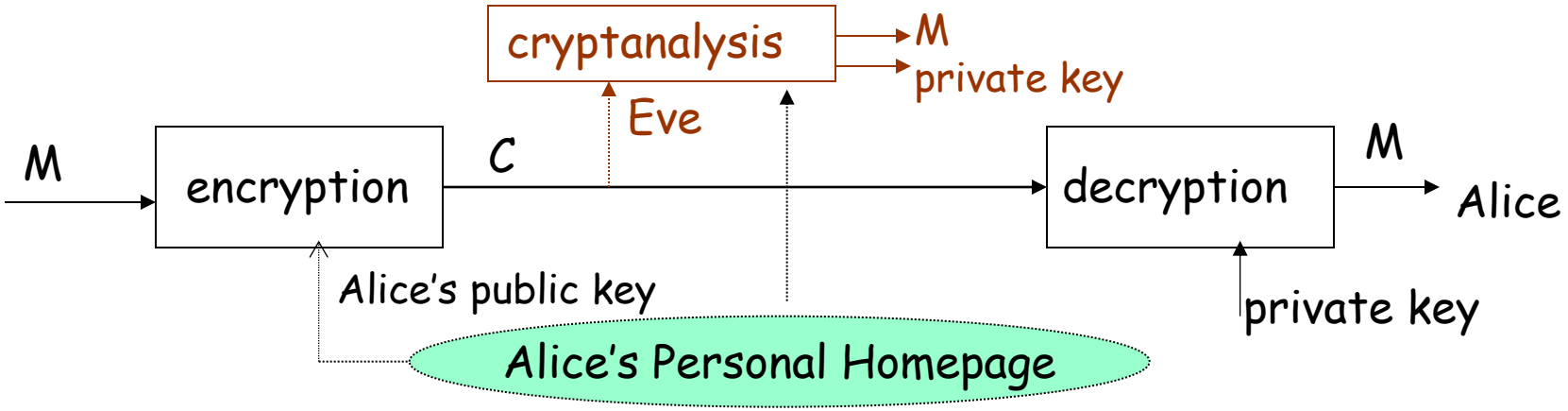
How do we verify a public key?



- How to verify *A*'s public key?

Digital Certificates

- Public key encryption: encrypt using receiver's public key
  - sender **has to be sure** that the public key used for encryption is indeed the receiver's public key
- Digital signature: verify a signature
  - Verifier **has to be sure** that the public key used for signature verification is indeed the signer's public key
- **How can the encryptor / verifier be sure that the public key is authentic?**



- How about posting the public key at a personal homepage?
- How about sending the public key to the encryptor / verifier using email?

Digital Certificates

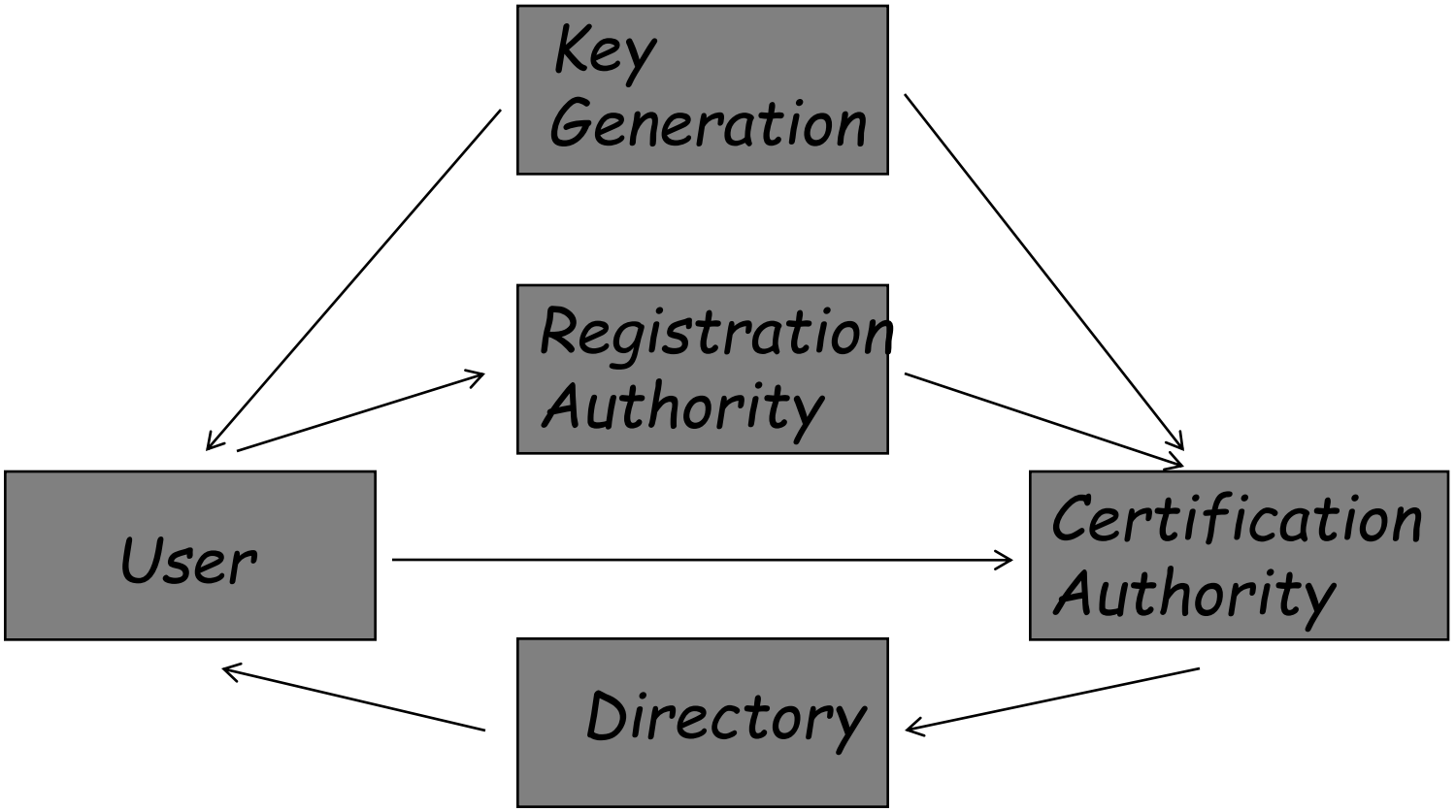
- How it works:
  - There is an entity called Certification Authority (CA) in the system
  - CA has a public key which is ASSUMED to be well known
    - e.g. built-in, preinstalled into all the web browsers/operating systems
  - CA issues a certificate to each public key owner
  - The certificate bears (1) the public key owner's identity, (2) the public key, (3) a validity period of the certificate and (4) the CA's signature
  - By using the certificate, the CA vouches that the public key in the certificate is owned by the public key owner.
  - The CA publishes a Certification Practice Statement (CPS) that specifies the policies (including liabilities) governing the use of the certificates issued.
- Only the CA can create a legitimate certificate
  - Only the CA can generate the signature in the certificate which requires the knowledge of CA's private key
- Anyone can verify the authenticity of the certificate using CA's public key

$Cert_A = (ID_A, PK_A, \text{expiry-date}, Sign_{CA}(ID_A, PK_A, \text{expiry-date}))$

The Certification Authority

- The "CA" is responsible for:
  - identifying entities before certificate generation
  - Generating user key or verifying user key
  - ensuring the quality of its own key pair,
  - keeping its private key secret.
- The CA, before generating a certificate, ought to check that a user knows the private key corresponding to its claimed public key.

Who is involved?



Some Remarks on Digital Certificates

- Certificate authority (CA) is *considered as* a Trusted Third Party (TTP) that issues and signs certificates
  - Verifying CA's signature in a certificate only verifies the **binding validity** between the public key and the identity in the certificate vouched by the CA
  - Verifying CA's signature does **not** verify the identity of the source that the certificate comes from!
    - E.g. Alice may receive Carol's certificate from Bob
  - Certificates are public!
  - Common format for certificates is ITU-T X.509.

X.509 certificates

- X.509 is a standard format for public key certificates (and CRLs).
- Standard first published in 1988.
- Latest was published in 2021.
- X.509 is part of the ITU-T Directory series of recommendations (ISO/IEC 9594).
- Certificate format specified in ASN.1.
- Note that the latest edition of X.509 covers 'attribute certificates' as well as public key certificates.

What's Inside a Certificate (X.509)

User Name
Certificate Version
Validity Period
Serial No
User's Public Key
Other user attributes
CA's name
CA's signature (of all the above)

e.g.

User Name (Common Name): www.hsbc.com.hk

Validity Period: Apr 16, 2010 – Apr 17, 2011

User's Public Key: RSA (2048 bits)

Modulus (2048 bits): 30 82 01 0a 02 82 01…

Exponent (24 bits): 01 00 01

CA's name (Issuer): VeriSign Class 3 Extended Validation SSL SGC CA

CA's signature (Certificate Signature Value): Size: 256 Bytes / 2048 Bits

There are many other attributes: Certificate serial no., certificate version number, HSBC public key algorithm, CA's signing algorithm, etc.

$$Cert_A = (ID_A, PK_A, \text{expiry-date}, \ldots, Sign_{CA}(ID_A, PK_A, \text{expiry-date}, \ldots))$$

Certificate Revocation

- A CA is responsible for the lifetime management of certificates, including renewal, update and revocation.
- Two methods for managing revocation:
  - use of Certification Revocation Lists, i.e. lists of revoked certificates, signed by CA
  - providing an on-line validation service.

CRLs

- Each CRL entry contains the serial number of the X.509 certificate being revoked.
- CRLs must be updated at regular defined intervals, enabling CRL user to verify that they are in possession of the 'latest' version.

Online certificate status protocol

- Online Certificate Status Protocol (OCSP) enables certificate status to be queried.
- OCSP may provide more timely revocation information than is possible with CRLs.
- Entity issues status request to TTP and suspends acceptance of certificate until TTP gives response.
- The protocol specifies data exchanged between entity checking certificate status and the TTP providing that status.

PKI

- Public Key Infrastructure (PKI) consists of all pieces needed to securely use public key cryptography
  - Key generation and management
  - Certification authorities, digital certificates
  - Certificate revocation lists (CRLs)
- No general standard for PKI
- We consider a few models of PKI

PKI Trust Models

Monopoly model

- One universally trusted organization is the CA for the known universe
- Big problems if CA is ever compromised
- Big problem if you don't trust the CA!
  - Should country X trust CA in country Y?
  - Where and who would this CA be?

PKI Trust Models

- Anarchy model
  - Everyone is a CA!
  - Users must decide which “CAs” to trust
  - Used in PGP (Pretty Good Privacy)
    - www.pgpi.org
  - Why do they call it “anarchy”?
    - Suppose cert. is signed by Frank and I don’t know Frank, but I do trust Bob and Bob vouches for Frank. Should I trust Frank?
    - Suppose cert. is signed by Frank and I don’t know Frank, but I do trust Bob and Bob says Alice is trustworthy and Alice vouches for Frank. Should I trust Frank?

PGP – Anarchy Model

- **Unstructured**
  - Suppose a public key is received and claimed to be Alice's.
  - The public key and Alice's identity are signed by some others (CAs). Each signature is considered as a certificate:

$$\text{Cert}_{\text{Bob}}(\text{Alice}), \text{Cert}_{\text{Carol}}(\text{Alice}), \text{Cert}_{\text{Dave}}(\text{Alice}), \text{Cert}_{\text{Eve}}(\text{Alice})$$

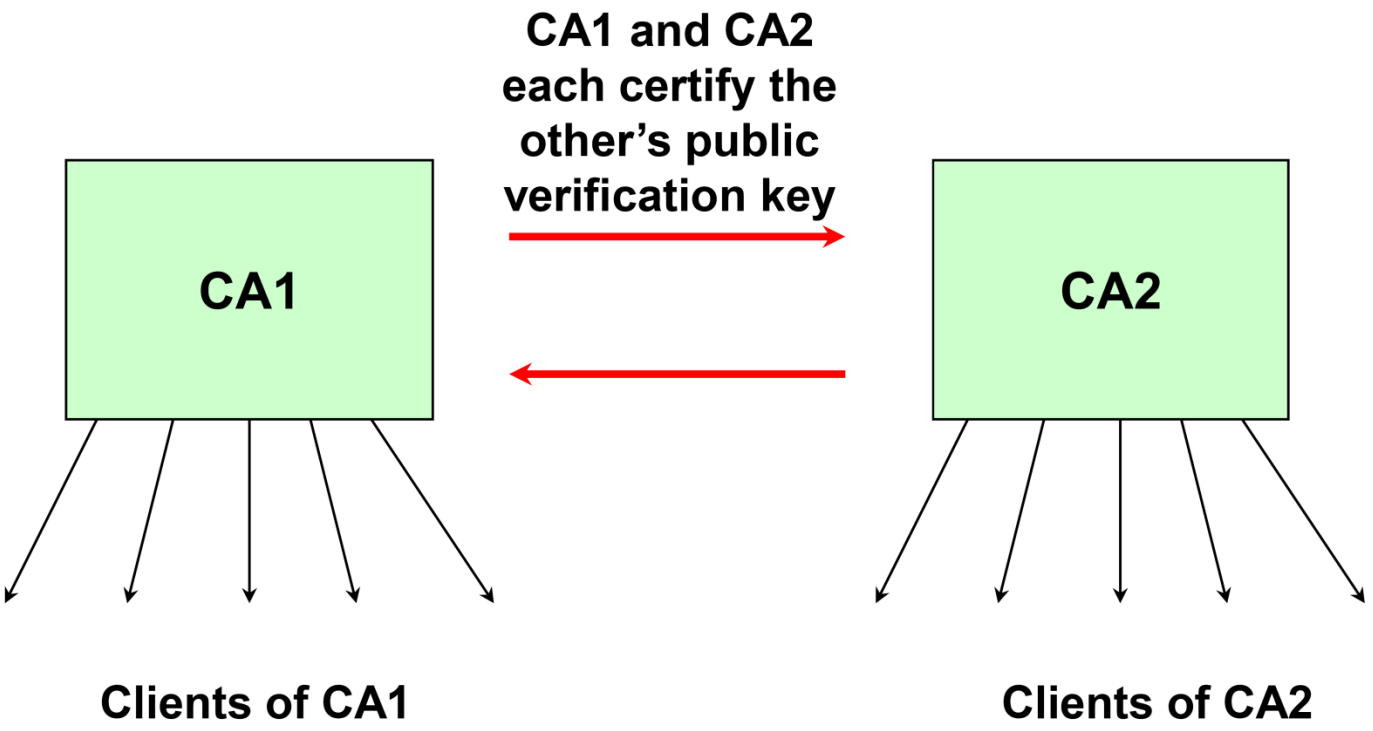
    Example: if my trust in certificates issued by Bob, Carol, and Dave (whose public keys I already have valid copies) are 1/2, 1/3, 1/3, respectively (and I don't have Eve's public key), then the above public key for Alice is considered as trust-worthy as $1/2 + 1/3 + 1/3 \geq 1$

  - Scalability weakness
    - Trust is not transferrable. Alice trusts Bob, Bob trusts Carol, does not necessarily mean that Alice trusts Carol.
    - It may be cumbersome for one to get adequate certificates so that one's public key can be trusted.

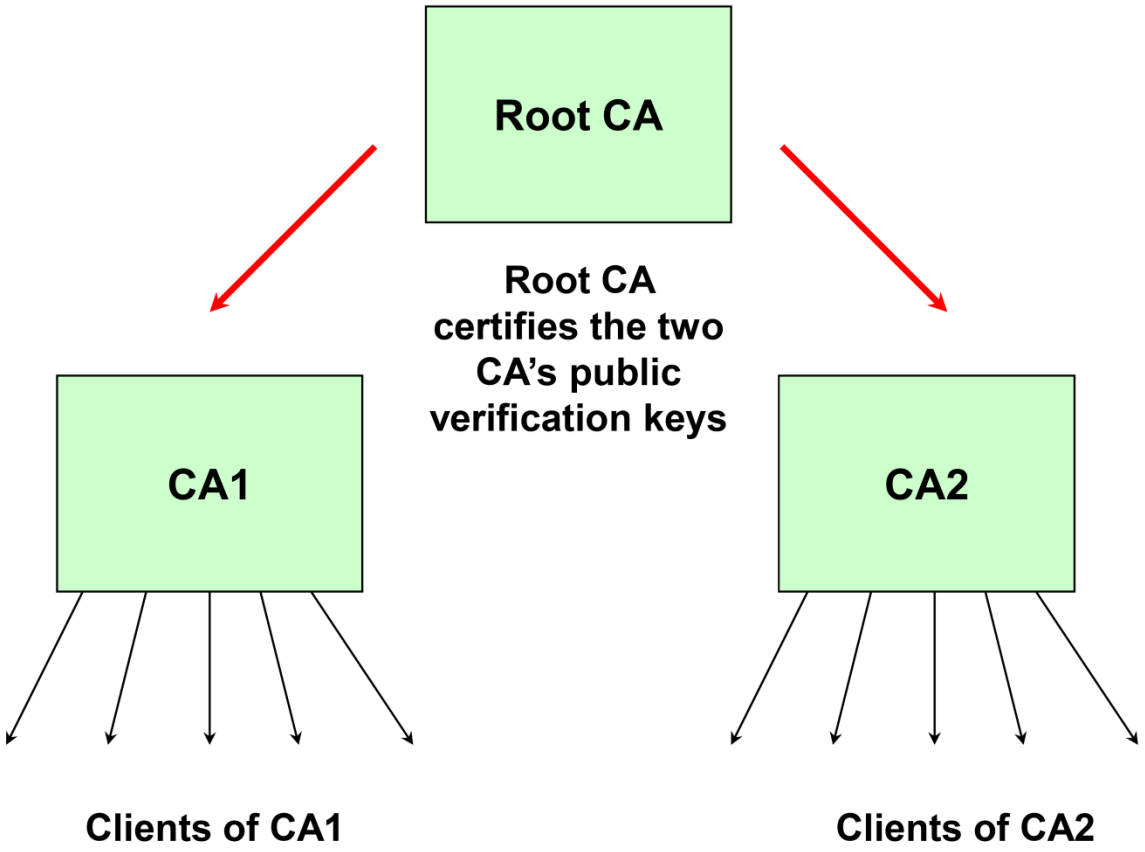
PKI Trust Models

- Structured
  - Multiple trusted CAs
  - Used today
  - Browser may have tens of root CAs' public keys build-in
  - User can decide which CAs to trust (by default, you trust what the browser said to trust)

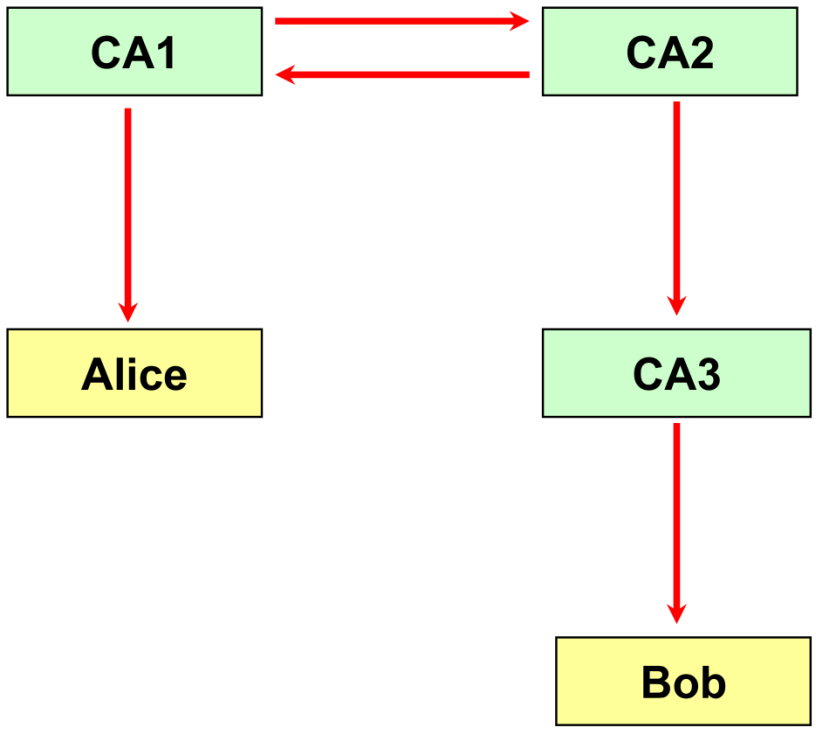
Cross certification



Certificate Hierarchy



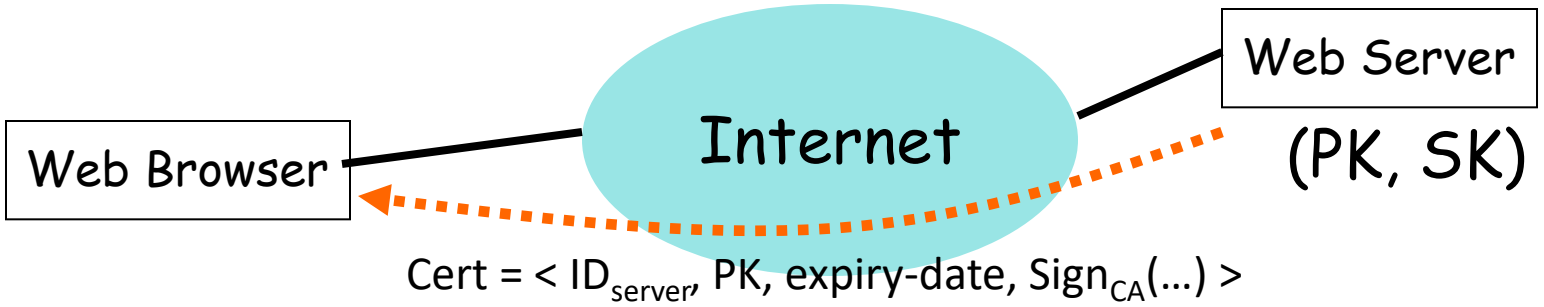
Certificate Chains



When Alice wants to check the authenticity of Bob's public key she must verify each link in the chain:

Public verification key of CA2	CA1
Public verification key of CA3	CA2
Public key of Bob	CA3

How to Use PKI – Secure Web Browsing



- The web browser has the CA's public key built in.
  - **The legitimacy of the web browser software becomes crucial for ensuring the security of digital certificates**
  - A certificate is **NO** more secure than the security of the web browser
- In practice, each browser trusts multiple CAs rather than just one
- Exercise: find out the number of CAs that your browser trusts

Payment Card Example

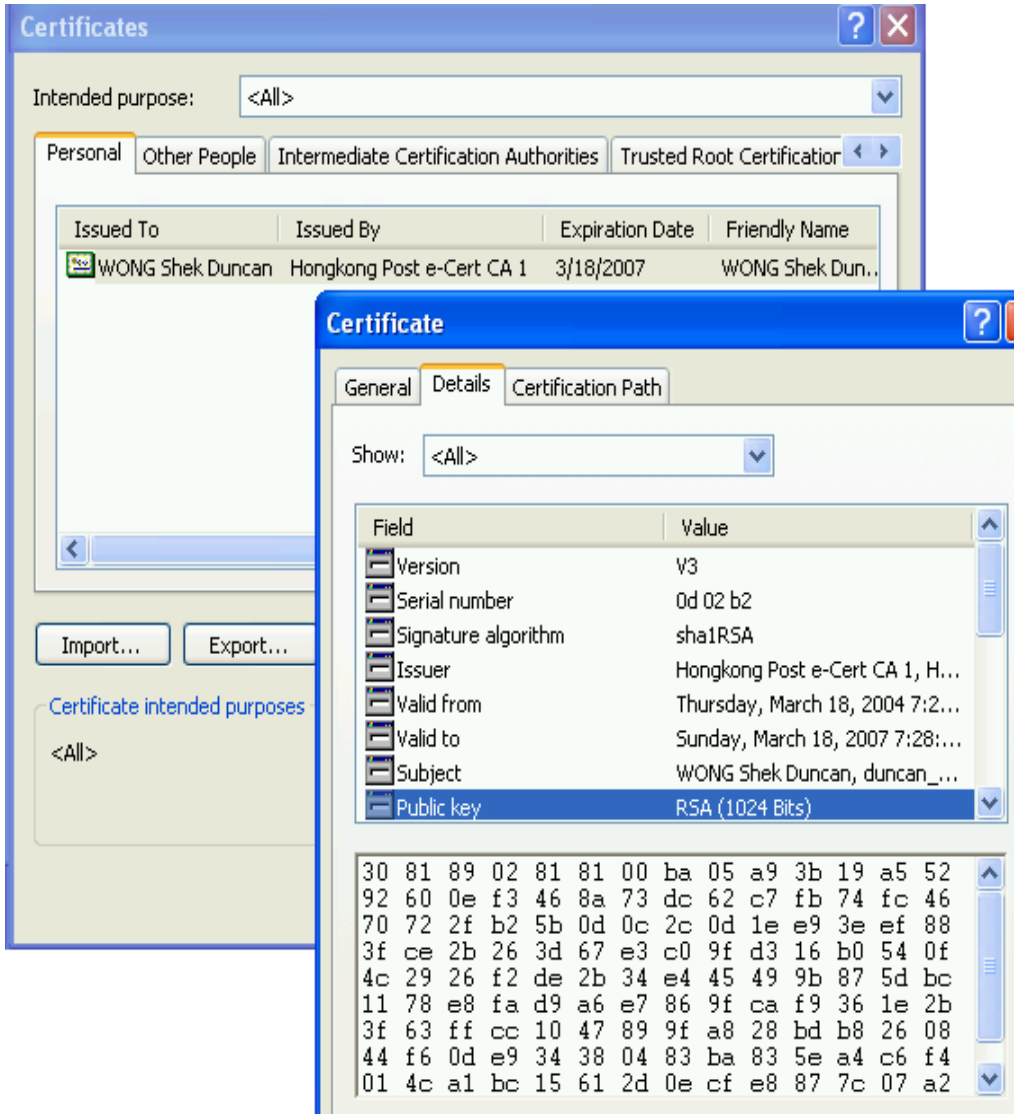
- EMV credit cards
  - Open payment (different card vendors and payment terminals different systems)
  - Authentication of card/transaction data
  - Card vendors not giving terminal vendor their master symmetric key!!! So what now?
- How can we authenticate if we do not trust others with our secret keys?
- Use public key crypto – use certificates!
  - Step 1: Card signs transaction data and also sends the card's certificate
    - Card certificate signed by card vendor
  - Step 2: Terminal has card company certificate, so verifies card certificate, then verifies card signature

PKI and e-Commerce

- A PKI can be used to ensure secure transactions on the Internet. This is especially important to foster e-commerce development.
- PKI implementation provides a solution for the business/legal aspects of electronic transactions.
- To bring such transactions to equivalent footing as traditional transactions some countries have established legislation governing electronic transactions.
- For example, in Hong Kong, the 'Electronic Transactions Ordinance' (http://www.ogcio.gov.hk/eng/eto/eeto.htm) was established in January 2000.
  - Give legal status to electronic records and transactions, digital signatures.
  - Give recognition to the first public CA, which went into operation in February 2000: Hong Kong Post e-Cert.

Personal Certificates

- Why want personal certificates?
  - send signed email messages
  - allow your peers to send encrypted emails to you
- Hongkong Post e-cert (Hong Kong's CA)
  - HK$ 50 for a year
  - E-cheque
  - Online (some banks login, tax returns, etc).



PKI Implementation Hurdles

- PKI does have great success, e.g. web security/TLS
- PKI is still not widely used for personal/business transactions. Good security and structural aspects but:
  - Not easy to understand by laymen.
  - Users have little incentive to get certificates as most applications are not PKI-enabled
  - Legal recognition largely untested
    - Very few real court cases (world-wide).
    - Must have certificate from recognised trust service provider
    - Very small number of these
- In Hong Kong, most of the applications support both personal certificate and password-based authentication methods.
  - Only two accepted CAs

Key management is boring!

Maybe, but it is very important!

- Without keys we have no security services

E-passport

- Sometimes need to design for interesting case
- E-Passports (governed by ICAO)
- Passive Authentication
  - Data signed
  - Who needs to verify?
- Basic access control
  - Verify symmetric key
  - Who needs this key?

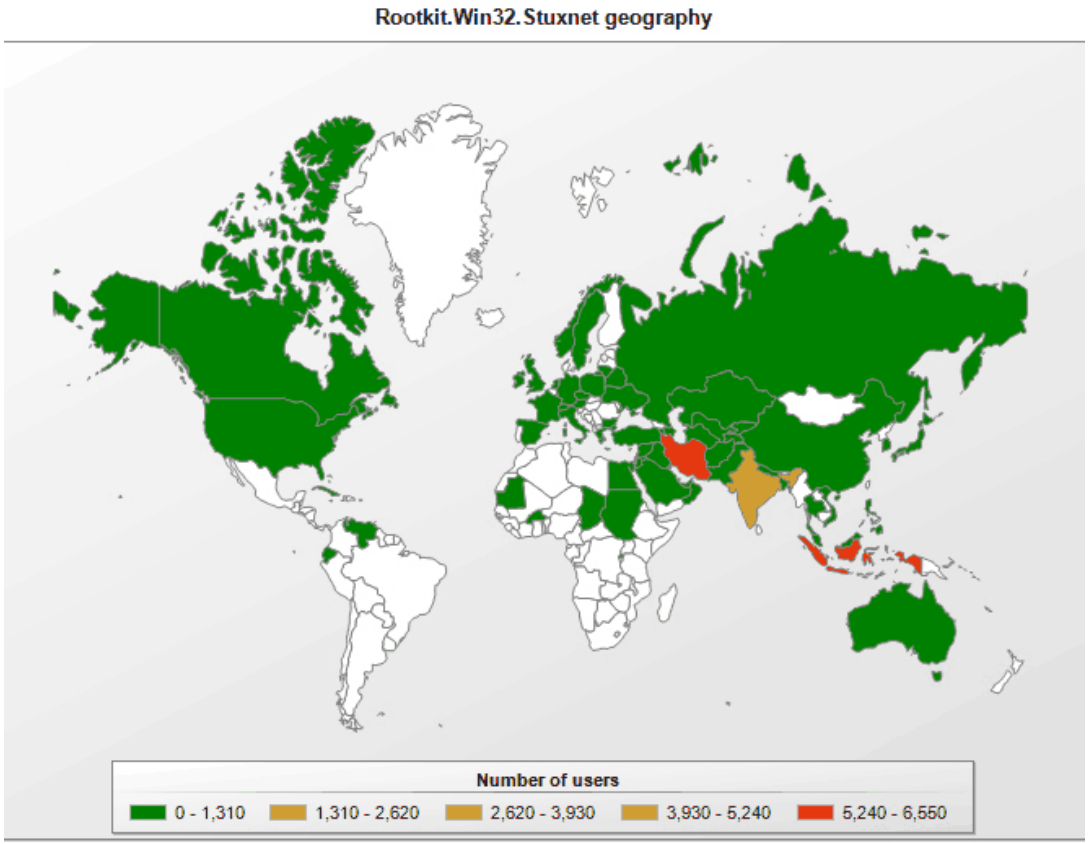
E-passport c'ntd

- Example of key management things going wrong!
- Basic access control key size
  - Theory: Serial no, birthday, expiry date = 50 bits
  - Reality: Predictable values, so closer to 25 bits
  - Had to fix: Alphanumeric random serial no
- Passive authentication verification
  - A few years after adoption not a lot of people were actually verifying the signature using country certificate

Advanced Threats

- Stuxnet was a multi-stage malware targeting intermediate systems (MS Windows,Siemens PCS7, WinCC) to reach its main objective (Siemens S7 PLCs).
- Stuxnet's delivery mechanism was based on the Microsoft Windows platform, but its primary objective was industrial control systems
- The control systems targeted were highly specific
- A number of zero-day exploits were deployed

STUXNET (1)



- Snapshot of STUXNET infection (Sept 2010)
  by University of Maryland.

Stuxnet (2)

- Stuxnet uses different mechanisms, including hiding itself on removable storage media (to get to air-gapped systems)
- A device driver is installed looking for files matching the characteristics of the Stuxnet payload.
  - What do you need to install driver on Windows?
  - Private key of trusted vendor
    - Jan-June 2010(Realtek), Verisign revokes certificate
    - July…(JMicron Tech)
- Core part of this issue: two compromised keys

The end!

Any questions…