Applied Cryptography Spring Semester 2023 Lecture 31

Kenny Paterson (@kennyog)

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

Overview of this lecture

- Introducing key management
- Generating keys
- Distributing keys
- Storing keys
- Controlling key usage

Additional slides (for information):

- Key change and key destruction
- Certificates and PKI
- Revocation
- Example PKIs

Introducing key management

What is key management?

- Key management refers to the secure administration of cryptographic keys.
- Encompasses technical, procedural, policy, management and human factors.
- Central in traditional areas of cryptographic deployment, e.g. banking and military/government communications.
- Often overlooked by developers when designing systems using cryptography.

Why key management?

Cryptography shifts the problem of securing data to the problem of managing the keys.

Anon

The importance of key management stems from the fact that no cryptographic system can perform its task properly if its keys are not properly managed.

A Key Management System (KMS) is any system for managing keys throughout their life.

Cryptographic key management

- The main requirement for the management of keys used with symmetric algorithms is that the keys remain secret.
- Relates to Kerckhoffs' principle: *"the security of a cryptosystem should not rely on the secrecy of the algorithms used, but rather in the secrecy of the key"*
- The main requirement for the management of keys used with public key algorithms are that the private keys remain secret and the public keys are authentic.
- An equally important requirement is to provide assurance of purpose of keys:

  A party in possession of a key should be confident that they can use the key for the purpose that they believe it to be for.

Cryptographic key management

Aspects covered by key management techniques:

- Key generation;
- Key storage;
- Key distribution and initialisation;
- Key usage and scope;
- Key replacement (key rollover/rotation);
- Key backup and recovery;
- Key revocation and destruction.

Generating keys

Key generation

Five methods to generate cryptographic keys:

1. Use of a truly-random or pseudo-random number generator to directly generate the key.

2. Establishing keys via a key agreement scheme such that the two parties wishing to communicate both contribute to the generation of keys.
   - For example, using Diffie-Hellman key agreement.

Key generation

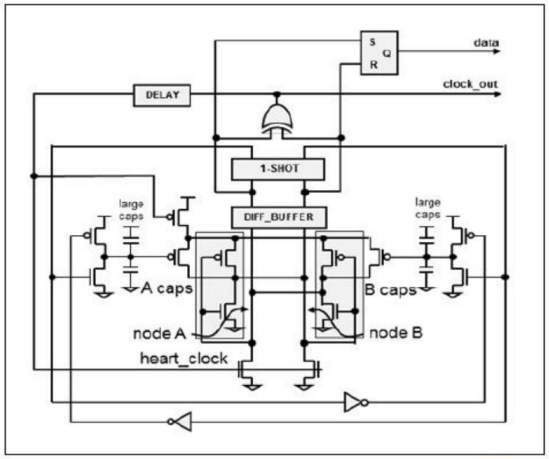
Both of these methods require access to a good source of randomness.

```c
#include <stdio.h>

int randomInt() {
    int n;
    return n;
}

```

Quantum RNG



Intel RD RAND generator

Cloudflare Lavalamps

Key generation (ctd)

3. Derive keys from a master key *MK*.

- The master key is not used to directly protect data, but rather to derive all subsequent keys as required.
- Typically achieved by using a Key Derivation Function (KDF), via:

$$K = \mathrm{KDF}(MK, \text{context})$$

where "context" is a context string shared by the entities who will subsequently use the key.

- The main requirement for a KDF is that the output key *K* should appear to be uniformly random, cf. PRF security.
- Example: EMV MAC keys used to protect transaction messages sent from the card to the issuing bank.
  - The keys *K* are statically embedded in cards at issuance time; the issuing bank derives them from *MK* and card PAN as needed.
  - The bank can then focus on protecting sa mall numbers of master keys, typically using special-purpose hardware.

Key generation by key derivation (ctd)

- Ad hoc methods: use a block cipher or a hash function:

$$K = E_{KM}(\text{context}) \quad \text{or} \quad K = H(KM \,||\, \text{context})$$

- May need to use chosen method several times on different context inputs in order to obtain enough bits.
- Security of first method relies on pseudo-randomness of block cipher, assuming unique context strings.
- Unclear exactly what property of hash function is being used here!

Key generation by key derivation (ctd)

HKDF: RFC 5869 – defines a general purpose KDF based on HMAC

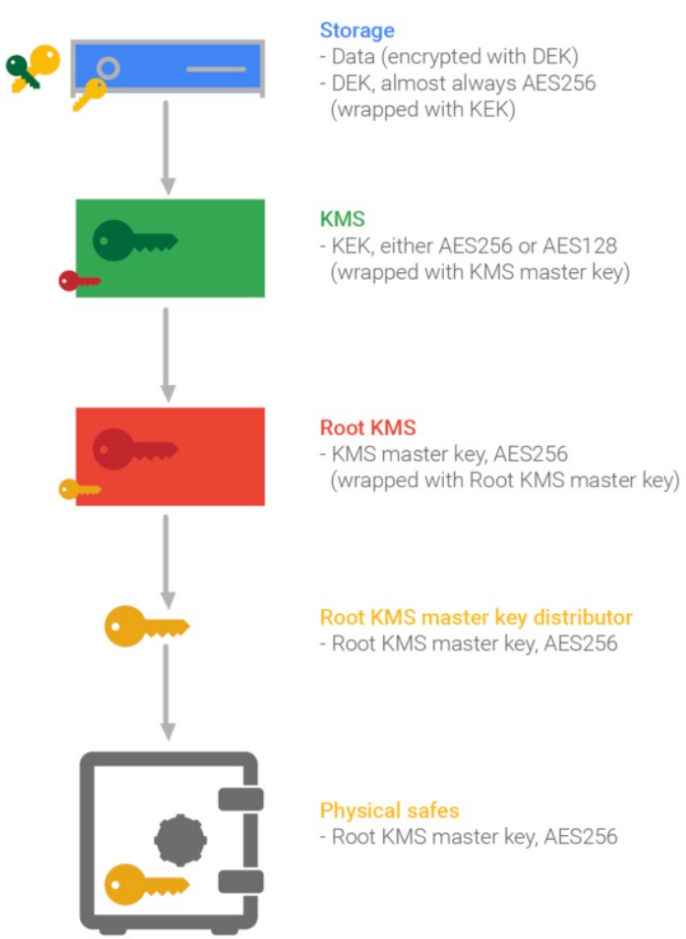
Uses an "Extract-then-Expand" construction, with optional, non-secret `salt` field, and context field called `info`:

```
PRK  = HMAC-Hash(salt, IKM)
T(0) = empty string (zero length)
T(1) = HMAC-Hash(PRK, T(0) | info | 0x01)
T(2) = HMAC-Hash(PRK, T(1) | info | 0x02)
T(3) = HMAC-Hash(PRK, T(2) | info | 0x03)
...
```

Pseudorandomness of outputs relies on HMAC being a *randomness extractor* and a PRF:

- Construction of `PRK` puts high-entropy secret material in second input.
- Construction of output blocks `T(i)` puts `PRK` in first input.
- HKDF introduced in: https://eprint.iacr.org/2010/264.pdf

Example: Google Cloud Storage Key Hierarchy



This type of key hierarchy enables lightweight *key rotation*:

- Generate new KEK.
- Re-encrypt all DEKs under new KEK.
- Encrypt new KEK under KMS master key.
- No re-encryption of bulk data required.
- Similar schemes used in AWS, Azure,…

https://cloud.google.com/docs/security/encryption/default-encryption

Key generation (ctd)

4. Derive keys from a previous key, using a scheme that ensures that no information is provided about the original key *if the newly generated key is compromised*.
   - Typically achieved achieved using a KDF, e.g.:

$$K_{i+1} = \mathrm{KDF}(K_i, \text{context})$$

   - Achieves a form of "forward security": compromise of the current key K_i does not compromise the security of data protected under previous keys K_{i-1}, K_{i-2},…
   - Careful synchronization of indices is needed.
   - A scheme like this is used in Signal, WhatsApp, etc where it is called *symmetric key ratcheting*.

Key generation (ctd)

5. Derive keys from PINs or passwords (or other inputs of dubious cryptographic strength).
   - Main issue here is that the resulting keys *K* are only as strong as starting values (PIN, password, etc).
   - Known plaintext or stereotypical plaintext format provides avenues for dictionary attacks on derived key *K*.
   - Use salting and iteration in special-purpose password hashing algorithm to slow down attacks.
   - PBKDF, scrypt, Argon2,…
   - Very common approach in “consumer-grade cryptography”.

Distributing keys

Key establishment

- Keys need to be exchanged between communicating parties
- This may involve a combination of manual and electronic means.
  - One of the main aims is to minimise manual involvement.
- Some types of key establishment schemes:
  - Master/session key (e.g. ANSI X9.17/ISO 8732)
  - Hybrid public key/symmetric key schemes (e.g. ISO 11770-3 / ANSI X9.44)
  - Unique key per transaction (e.g. ANSI X9.24)
  - Diffie-Hellman key exchange (e.g. ANSI X9.42)

The ETCRRM
Image © NSA

Hybrid public key/symmetric key scheme

- Use a public key encryption scheme to encrypt a symmetric Key *K.*
- Technically, only a KEM is needed rather than a full-fledged PKE scheme.
- Now only public keys for encryption need to be distributed.
  - No need for secrecy of public keys, but authenticity of public keys is required.
  - Authenticity often provided via the use of a Public Key Certification Authority (CA).
  - Once upon a time, was widely used in the TLS protocol (largely phased out in favour of Elliptic-Curve Diffie-Hellman key exchange).

Unique key per transaction

- A number of schemes exist that can provide automatic update of keys with every usage of the keys, e.g. per transaction in a financial system.
  - This way, there are no "static" keys in the system.
  - Makes multi-sample side-channel attacks more difficult to carry out.
- For example: derive new key from current key and transaction counter using a PRF or a hash function, cf. Method 4 above.
- Particularly useful in retail environments, where relatively insecure payment terminals are used.
- Possible synchronisation issues.

Diffie-Hellman Key Exchange

- Public key method for agreeing on a shared secret (which can be used as a session key, for example).
- Introduced in famous 1976 paper that launched public key cryptography.
- Susceptible to MITM attacks unless values used in the method are properly authenticated.
- Discussed extensively in earlier lectures, and will be covered again in next lectures on Authenticated Key Exchange.

Storing keys

Key storage

Keys need to be stored securely.

Storage options include:

- Inside a tamper-resistant hardware security module (TRSM or HSM).
- Inside a smart card or other personal token.
- Outside a TRSM encrypted with a key encrypting key.
- Outside a TRSM in the form of multiple key components.

However, the reality is that, in many systems, keys are stored in memory that is afforded only the usual OS protections.

- e.g. server private key in TLS key exchange.
- Heartbleed as an example of the dangers of this approach.
- https://dl.acm.org/doi/10.1145/2663716.2663755

Hardware Security Modules

Secure key storage usually requires the use of a tamper-resistant hardware security device, such as a hardware security module (HSM).

Usually some form of local master key (LMK) is stored inside the device.

- Other keys, encrypted under the LMK, can be held outside the HSM, but submitted to the HSM when needed.
- Using the LMK, the HSM can decrypt to recover keys, and perform actions with them.
- In some cases, all the keys may be held inside the security module.

Typical HSM facilities

- Functions:
  - Key management support,
  - Other cryptographic security processing.
- Physically secure.
- Tamper resistant – meeting, for example, US FIPS 140-2 standard
- Deployed as a peripheral to a host computer
- Security through restricted interface
  - Access control for sensitive functions
- Highly configurable.
- Expensive!

Tokens

- Hardware tokens
  - PC cards (PCMCIA) – e.g. Fortezza card
  - Smart cards – SIM/USIM/EMV and other payment cards
  - Universal Serial Bus (USB) tokens – e.g. i-Key
  - Cryptographic tokens
  - See https://www.imperialviolet.org/2017/08/13/securitykeys.html for an interesting analysis of what was available on the market in 2017
  - Increasing usage due to growth of multi-factor authentication and due to cryptocurrency custody solutions.
- Software tokens
  - PSE files – PKCS#12
  - OpenPGP key formats

Key storage in software

Hidden in software, "in the clear"

- Cheap but dangerous!
- Developer who designs software knows where they are.
- An attacker who has access to two different versions (with different keys) can run a comparison.
- Can try to obfuscate relevant code and key data.
- White-box cryptography.

Controlling key usage

Key usage

Principle of Key Separation

A cryptographic key should only be used for its intended purpose

- Assumes the purpose is defined and limited!
- A given key should in general not be used in more than one cryptographic algorithm.
- For example, a key used for message authentication or entity authentication should never be used for data encryption.
- Instead, it is recommended that the many different cryptographic keys are generated independently.

Principle of Key Separation

- Main reason for key separation: the interaction between algorithms may result in security weaknesses.
  - Example 1: using the same key for CBC mode and CBC-MAC.
  - Example 2: using the same key pair for RSA encryption and signatures.
- Lack of appropriate key separation can lead to significant vulnerabilities in practice. See, for example:
  - https://www.kopenpgp.com/
  - https://mega-awry.io/
- Keys for encryption and authentication may have totally different lifecycles, and internal policy may have conflicting requirements concerning storage, backup, destruction.
- Using different keys for different purposes also limits the damage that may arise from key compromise.

Controlling key usage

- Because of the key separation requirement, techniques may need to be put in place to enforce the correct usage of cryptographic keys.
- This is often done by associating attributes restricting their use and other operational information to the key.
- These attributes may include ownership, validity, identifier, intended use and algorithm associated with the key.
- Requires a key hierarchy and:
  - A labeling scheme.
  - A binding method.
  - An enforcement method.
  - The binding and enforcement are often provided via HSM functions.

Key management standards

There are *many* standards and guidance relating to key management.

For example:

- ANSI X9.17 / ISO 8732 (withdrawn)
- ANSI X9.24 Retail Financial Services Symmetric Key Man.
- APACS 40 & APACS 70 (UK)
- ISO 11568 Key management (retail)
- ISO 11770 Security techniques – key management
- ISO 15782 Certificate management (financial services)
- RFC 4107 Guidelines for cryptographic key management
- NIST SP 800-57 Recommendations for key management
- IEEE P1619.3 Key management standard working group
- OASIS Key Management Interoperability Protocol

NIST Key Management Docs:

http://csrc.nist.gov/groups/ST/toolkit/key_management.html

Homework

- Good coverage in chapters on key management in "Everyday Cryptography" by Keith Martin.
- Second assessed lab assignment is due this Friday!
- Exercise class this week on signatures and ECC.

Additional Slides

Key change and key destruction

Key change

- Planned key updates arise from:
  - End of key lifetime.
  - Data limits on key usage exceeded.
    - Example: TLS 1.3 data limits derived from analysis of AES-GCM and ChaCha20Poly1305 security proofs.
    - See https://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf and https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-aead-limits-02
- Unplanned key updates may arise from:
  - Key compromise.
  - Security vulnerability discovered that could lead to key compromise, or could already have resulted in key compromise (e.g. Heartbleed).
  - Unexpected departure of an employee.

Impact of key change

- Minimum impact: a new key has to be generated and established.

- Impact could include:
  - costs of distributing new smart cards
  - costs of investigation
  - costs of changing key management system
  - costs relating to repairing activities conducted using compromised keys
  - damage to reputation
  - loss of confidence in system

Key destruction

- Must be done using a secure mechanism for destruction of sensitive data
  - when key expires
  - when key is withdrawn
  - at end of a key archival period

- Usually involves over-writing memory or storage medium, but may require physical destruction of, e.g., key storage devices.

Key management – policies, practices and procedures

- Key management policies
  - Define overall strategy at an organisational level.
- Key management practices
  - Describes tactics used to achieve the policy.
- Key management procedures
  - Step-by-step tasks to implement specific practices.

Certificates and PKI

Public Key Management System

- A framework established in which asymmetric keys (private and public) are created, stored, distributed and managed.
- PKI (Public Key Infrastructure):
  - Often misused to denote public-key cryptography itself.
  - Strongly associated with public-key management based on certificates (not the only way)
  - All cryptosystems require a key management infrastructure (but we tend not to talk about SKIs)
- Public Key Management Schemes are used to support a number of security functions:
  - Encryption
  - Authentication
  - Authorization
  - Access Control

Using a trusted directory

We could use a **trusted directory** that lists

- All public keys
- All related data (next to each key)
- Original concept from Diffie-Hellman paper.

Public key certificates

A **public key certificate** is a set of data that binds an identity to a particular public key. The four **core** pieces of information that are contained in a public key certificate are as follows:

Name of owner

- Could be a person, device, or even role.
- Should uniquely identify the owner within the environment in which the public key will be employed.

Public key value

Validity time period

- Identifies date and time from which the public key is valid and the date and time of its expiry.

Signature

- Creator of certificate digitally signs all data that forms the public key certificate. This binds the data together and acts as a guarantee that the creator of the certificate believes the data to be correct.

X509 v3 public key certificates

Version	This specifies the X.509 version being used (in this case v3).
Serial Number	A unique identifier for the certificate.
Signature Algorithm	The digital signature algorithm used to sign the certificate.
Issuer	The name of the creator of the digital certificate.
Validity	The dates and times between which the digital certificate is valid.
Subject	The name of the owner of the digital certificate.
Subject Public Key Info	The actual public key and the identifier of the public key algorithm associated with it.
Issuer Unique ID	An optional identifier for the creator of the digital certificate.
Subject Unique ID	An optional identifier for the owner of the digital certificate.
Extensions	A range of optional fields that include: ■ a key identifier (in case owner owns more than one public key) ■ key usage information that specifies valid uses of key ■ the location of revocation information ■ identifier of the certificate policy ■ alternative names for the owner

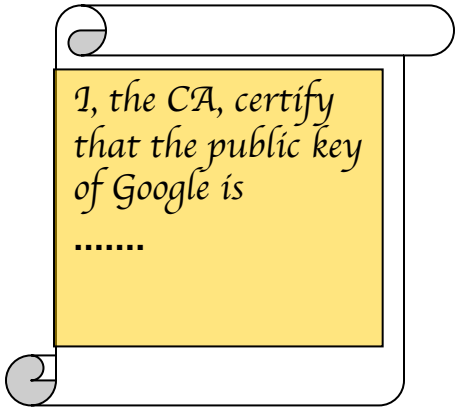
Certificate Authorities

A creator of a digital certificate is normally referred to as a **Certificate Authority** (or **CA**).

1. The CA takes responsibility for ensuring that the information in a certificate is correct. The CA creates (or issues) the public key certificate to the owner.
2. Whenever anyone has need of the owner's public key they request a copy of the public key certificate – possibly from central server or the owner.
3. The recipient of the public key certificate checks that the certificate is in order (including verification of the CA's signature), and if they are happy with it, then they are free to use the public key contained in the certificate.

Meaning of certificates

By digitally signing the information in the public-key certificate, the Certification Authority is effectively making the statement:

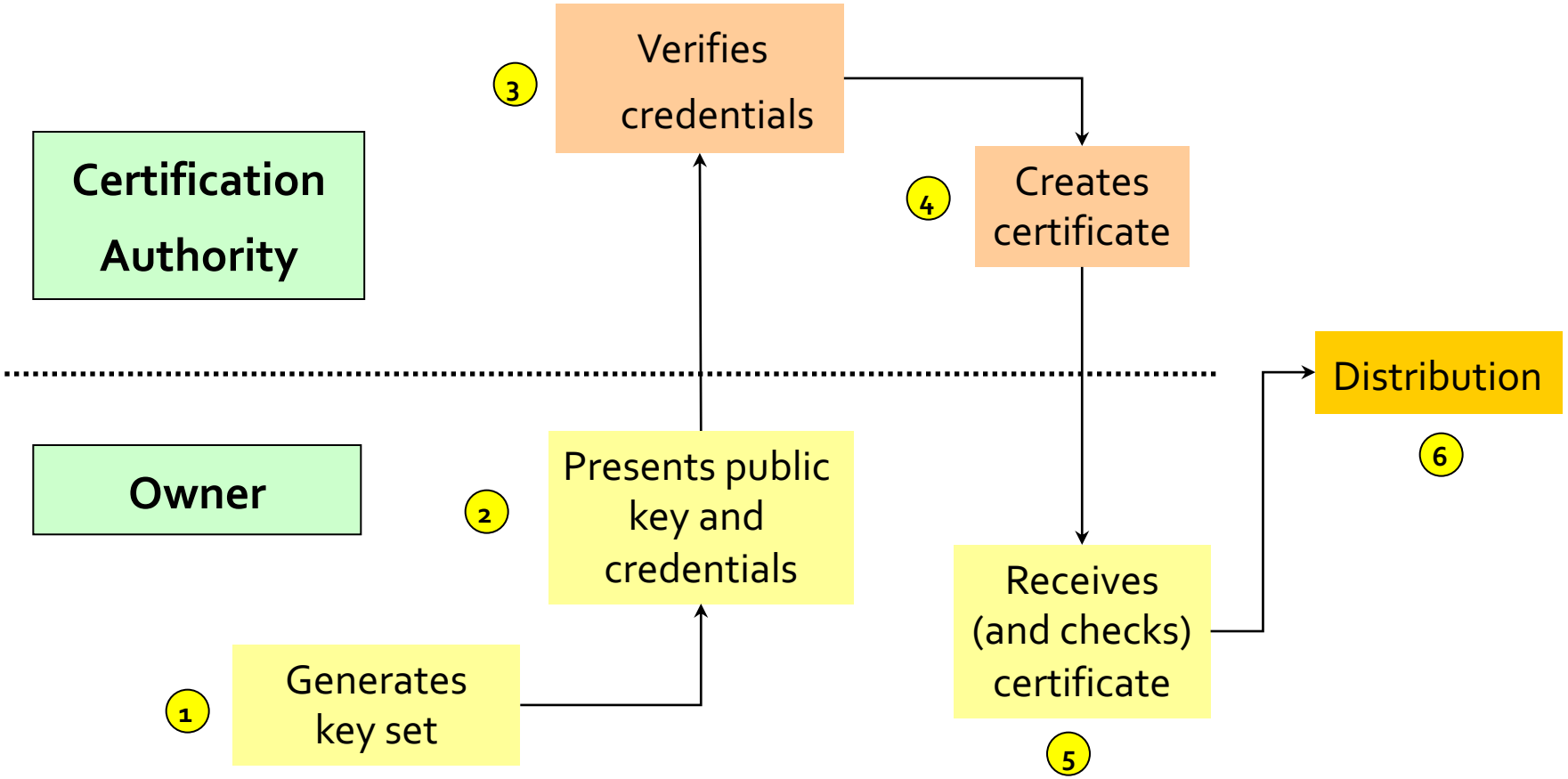


Trusting a digital certificate

There are three things that the recipient (the relying party) needs to do in order to be satisfied that the public key certificate is in order:

1. The recipient needs to be able to trust (directly or indirectly) the CA to have done their job correctly and to have gone through some process to verify all the fields of the certificate.
2. The recipient needs to have access to an authentic public verification key for the CA in order to verify the CA's digital signature on the certificate.
3. The recipient needs to check all the fields in the certificate. In particular it must check that the certificate is valid, that it applies to the correct owner and that the other fields are all satisfactory.

Example certificate issuing process



Registration

The stage of the certification process at which the owner presents their credentials to the CA for checking is a vital stage in the entire certification process.

In many application environments a separate entity known as a Registration Authority (RA) performs this operation.

There are two arguments for separating the roles of CA and RA:

1. Most of the functionality of a CA can be essentially performed by a computer, whereas for many applications the role of the RA requires human intervention.
2. Checking the credentials of a certificate applicant is often the most complex part of the certification process. There is thus a strong argument for distributing the registration activities across a number of "local" RAs.

Levels of certificate

- Many CAs issue different types of certificate (sometimes referred to as **levels** of certificate) depending upon the thoroughness of the process used to identify the owner.
- These levels often correspond directly to the quality of the credentials by which the owner was identified when the certificate was registered and by the rigorousness with which they are checked.
- Certificates of different levels may then be used in different types of application, e.g. identity verification versus code signing.
- The liability associated with a certificate is likely to be different for different levels of certificate.
- The CA/Browser Forum publishes "baseline requirements" for certificates and guidelines for Extended Validation certificates.

Proof of possession

- The owner should show that it actually is in possession of the private key corresponding to the public key in the certificate.
- This process is referred to as demonstrating Proof of Possession.
    1. For signature verification keys, this can be done by signing a certificate request message using the private key.
    2. For encryption keys, this can be done by the CA encrypting the issued certificate under the public key that was certified.
- Frequently done using the first method, even for non-signing key pairs, formally violating the key separation principle!
- Technically, not a proof in the strict sense typically used in cryptography.

Revocation

Revocation

We must consider how to handle certificates that need to be "withdrawn" before their expiry date. This process is usually referred to as certificate revocation.

There are two main approaches used in practice: CRLs and OCSP.

Certificate Revocation Lists (or CRLs)

- CA maintains a list of certificates that have been revoked, the CRL.
- CRLs obviously need to be maintained carefully.
- CRLs need to be signed by the CA.
- Updated CRLs need to be distributed at regular intervals.
- CRLs may grow in size over time (can eventually remove certs which have expired anyway).

Revocation

Online Certificate Status Protocol (OCSP)

- The CA maintains an online database containing the status of certificates it has issued; relying parties query the database to check status of individual certificates.
- Requires online access for relying parties.
- May consume huge bandwidth at CA, mitigated by using CDNs.
- Privacy-invasive, since CA learns which certificates are being queried by users in the system.
- **OCSP stapling**: owner of cert attaches a **recent** OCSP response from the CA when sending its certificate to a relying party.

Revocation in practice

In practice, and because of security failures, CRLs and OCSP have been augmented by other approaches for the PKI that is used to support SSL/TLS on the World-Wide Web.

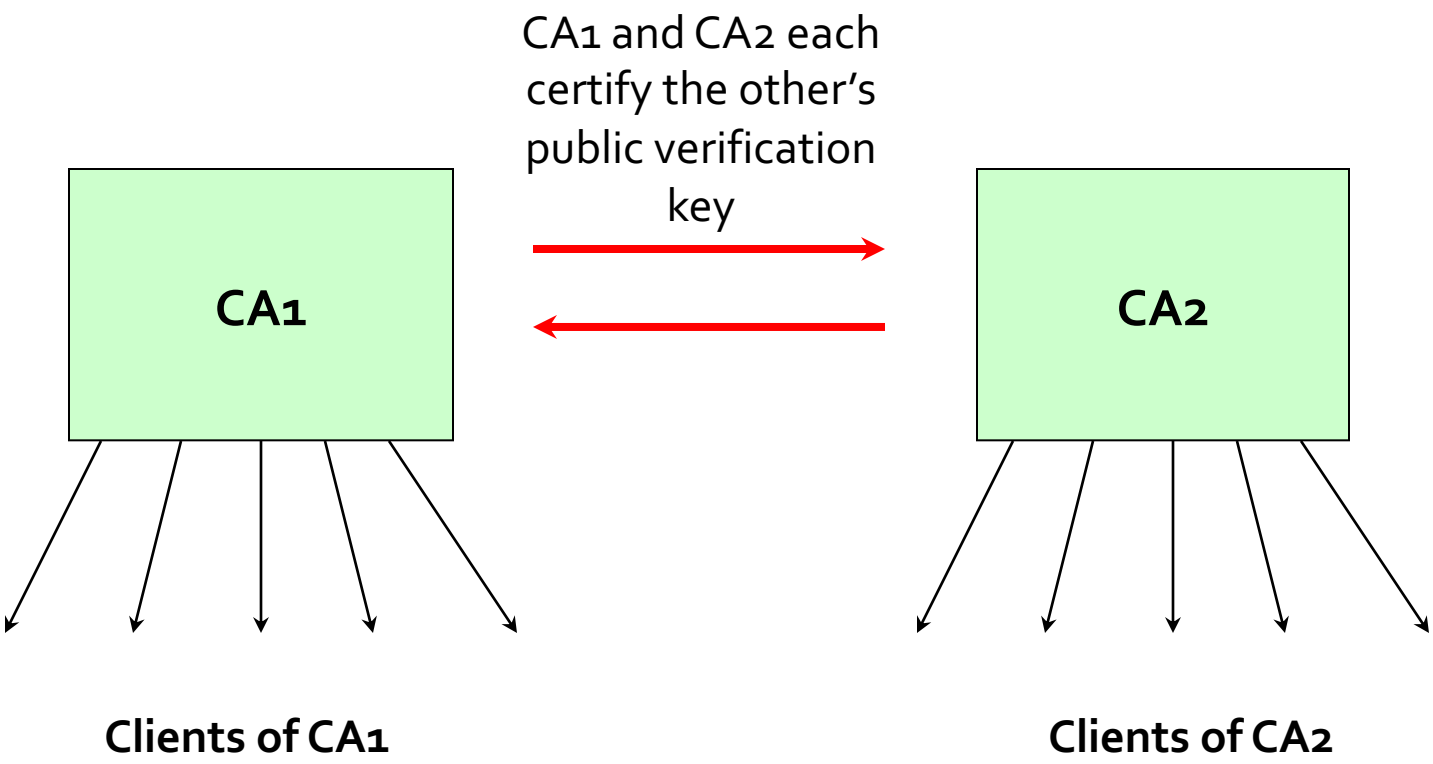
Certificate transparency

- Google-led effort, specified in RFC 6962.
- Builds a publicly auditable log of observed certificates issued by browser-trusted CAs.
- Essentially, a Merkle hash tree of all certificates.
- Certificate/domain owner can check log for wrongly-issued certificates and pin-point misbehaving CA.
- More details at http://www.certificate-transparency.org/

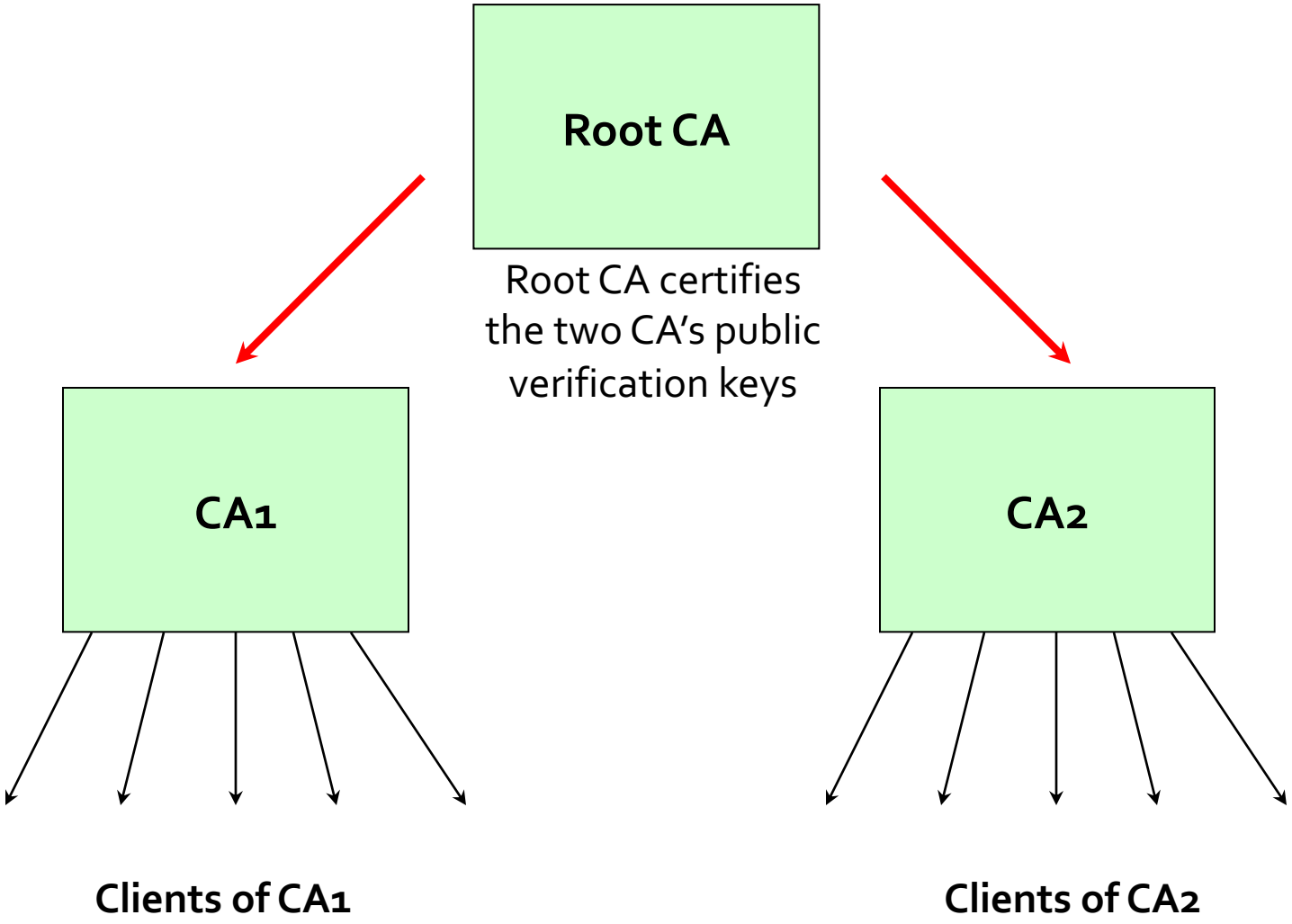
Joining CA domains

- An owner of a public-key certificate has by necessity placed some trust in the CA who has issued this certificate.
- For large (open) systems, a public-key certificate owner may want to
  - Communicate with relying parties who do not have a business relationship with the owner's CA; and
  - Rely on certificates that were not issued by the owner's CA.

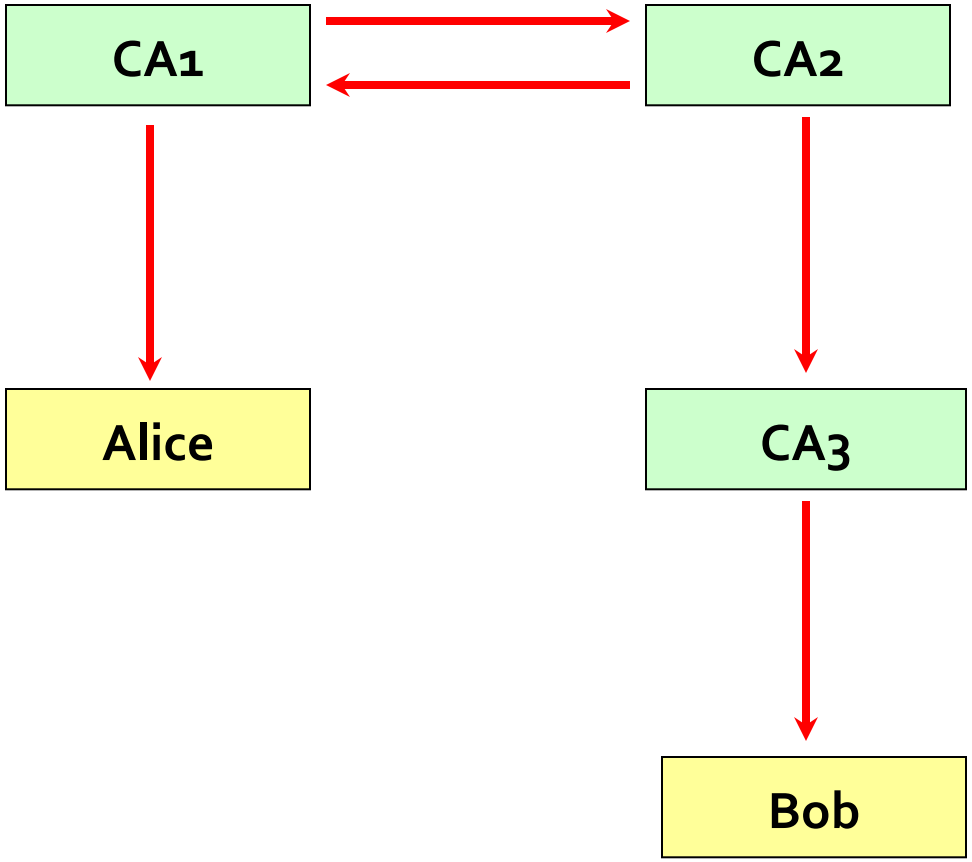
Cross-certification



Certificate hierarchies



Certificate chains



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

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

Example PKIs

An Example PKI: the "Web PKI"

- Browser vendors place CA root certificates in browsers.
- These are then used as roots of trust for chains of certificates.
- Lowest level of chain provides binding of domain name to public key.
- Certificate chains and public keys are exchanged during SSL/TLS Handshake Protocol.
- Some of the many problems:
  - Ineffective revocation methods implemented in browsers.
  - Large number of root CAs, many of dubious quality: Diginotar, Turktrust, CINIC incidents (see http://arstechnica.com/security/2015/04/google-chrome-will-banish-chinese-certificate-authority-for-breach-of-trust/).
  - "Browser trusted" is absolute: presence or absence of browser lock icon.
  - OEMs mess with root certificates to install spyware/ad injection software: SuperFish, Dell incidents (see http://arstechnica.com/security/2015/11/dell-apologizes-for-https-certificate-fiasco-provides-removal-tool/).

An Example PKI: the EMV PKI

- EMVCo: a company jointly owned by American Express, Discover, JCB, MasterCard, UnionPay, and Visa.
- Maintains specifications enabling interoperation of chip-based payment cards.
- Runs a substantial PKI.
- Each payment card is equipped with a signing key (currently RSA) and an issuer certificate for the corresponding public verification key.
- Card receives a challenge from the payment terminal to prove identity during transaction (DDA/CDA), transfers issuer certificate and signed response to terminal.
- Terminal verifies signature and certificate using local copy of issuer public key.
- Nice overview of the entire system from a key management perspective: https://www.cryptomathic.com/hubfs/docs/cryptomathic_white_paper-emv_key_management.pdf