



1495

UNIVERSITY OF
ABERDEEN

CELEBRATING
525 YEARS
1495 – 2020

ABERDEEN 2040

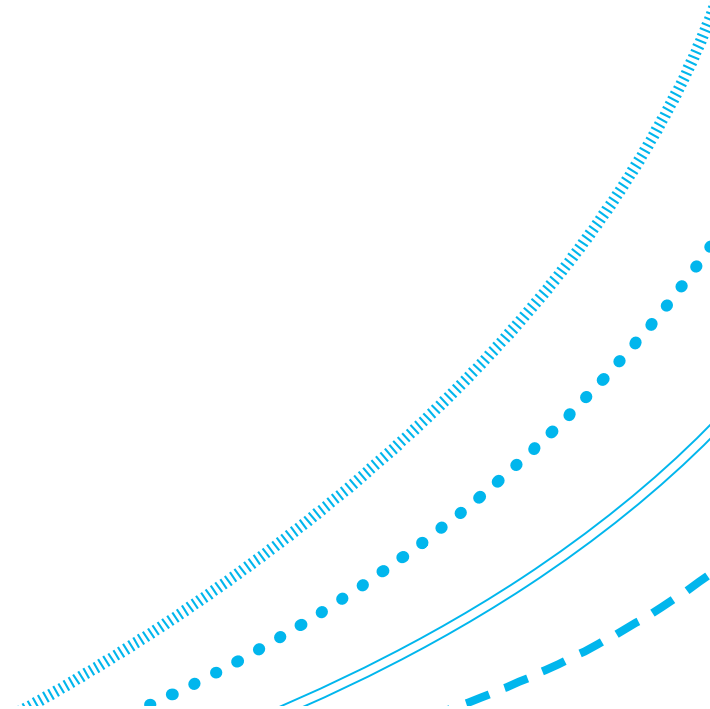
Network Security Technology

Certificates

September 2025

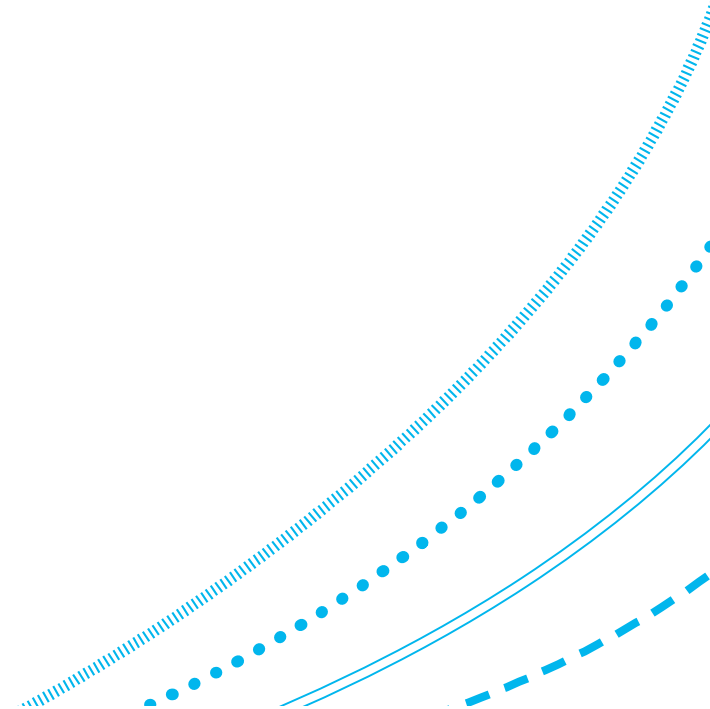
Outline of lecture

1. Binding keys to identities and certificates.
2. Public key infrastructure.
3. X.509 and PKIX.



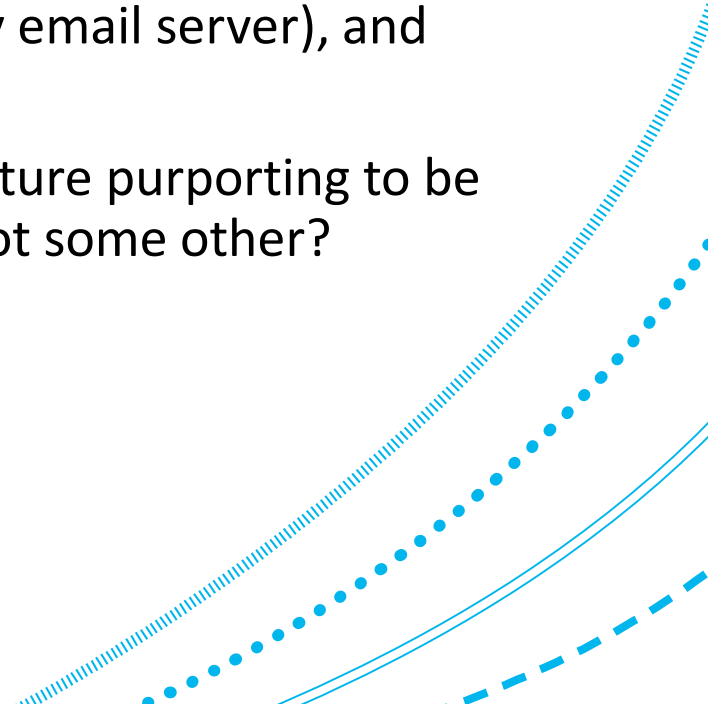
Attack on public key and binding

1. How can we have assurance that a public key, purported to come from some entity, really *does* belong to that identity?
2. We need to **bind** the identity to the public key.



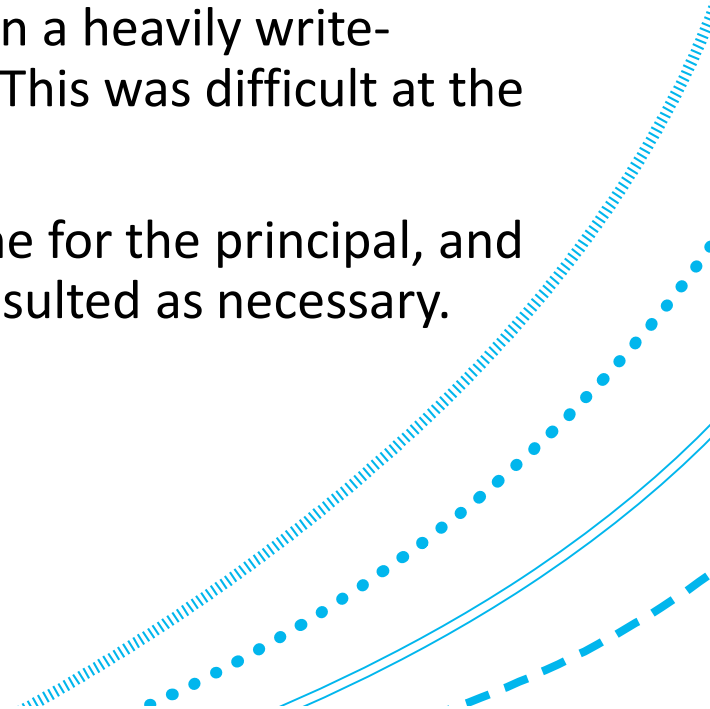
Attack on public key and binding: an example

1. I may publish my public key on my webpage in order for others to send me an encrypted message in email.
2. An attacker who can access and alter my webpage could change the public key to their own, intercept the emails (or get them from my email server), and then decrypt the content.
3. Similarly, if a public key is used for verification of a signature purporting to be by some entity A, how do we really know it is A's and not some other?



Possible binding solutions

1. If the principal (entity) needs to be physically present, then you can have them carry the key around on a tamper-resistant smart card with protection by a password/PIN etc.
2. The original proposal was in fact to have bindings listed in a heavily write-protected file that could easily be accessed for reading. This was difficult at the time it was proposed (1976).
3. Use a **(public key) certificate** that contains a unique name for the principal, and the public key. Have such certificates transmitted or consulted as necessary.



Certificates: main idea

1. A **(public key) certificate** is a signed message containing:
 1. The entity's identity.
 2. The value of its public key.
 3. Other stuff, e.g. expiry date.
2. Analogy: Can loosely think of them as letter of introduction for the entity, signed by someone.



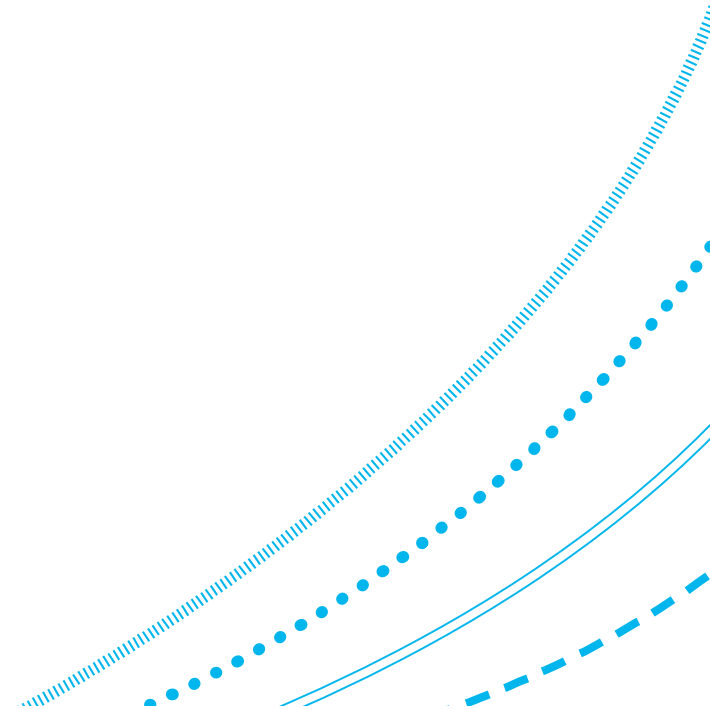
Certificates: notation

1. In abstract terms, certificates have the form:

$$Sig_{CA}(A, v_A)$$

where:

- Sig_{CA} is a signature by the signer,
- A is the entity named, and
- v_A is the public key of A .



Certificates: an example

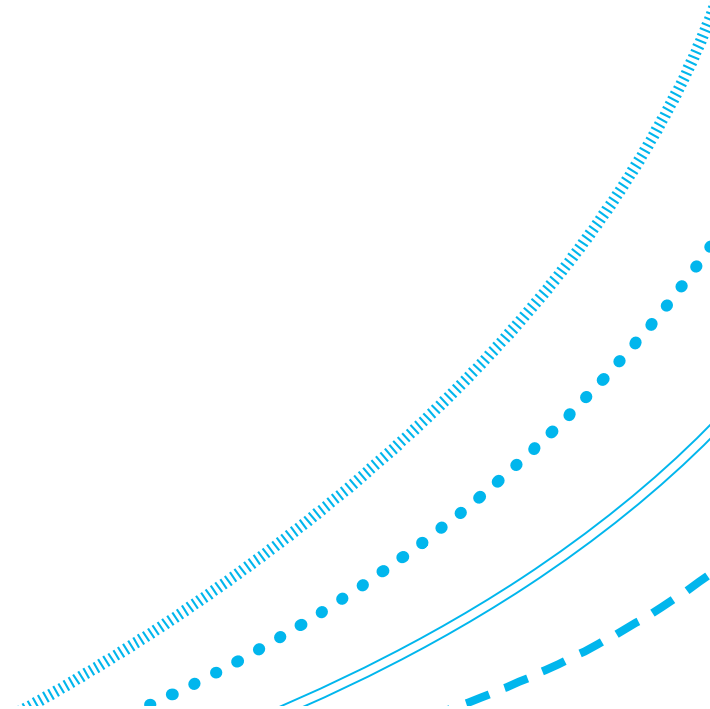
Certificate:

Key: v_A

Identity: A

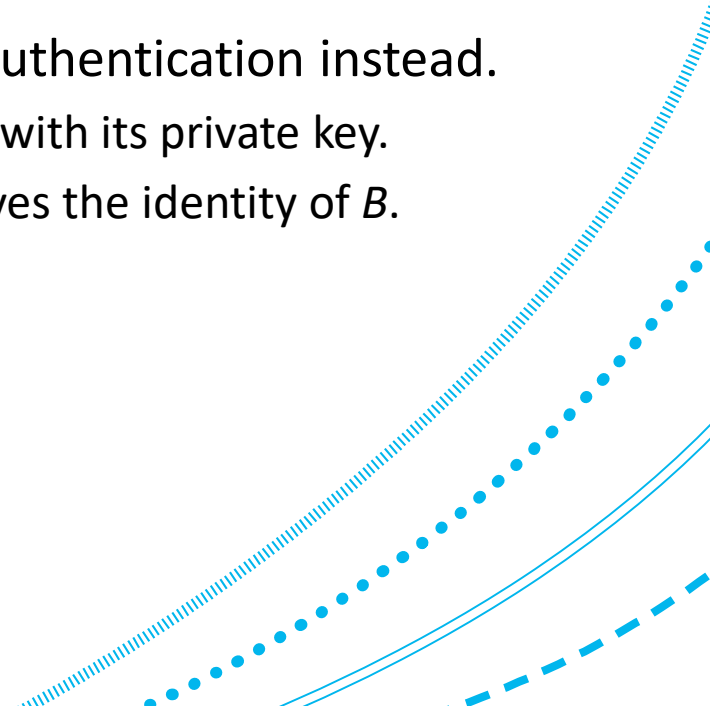
Signed by: CA

$$Sig_{CA}(A, v_A)$$



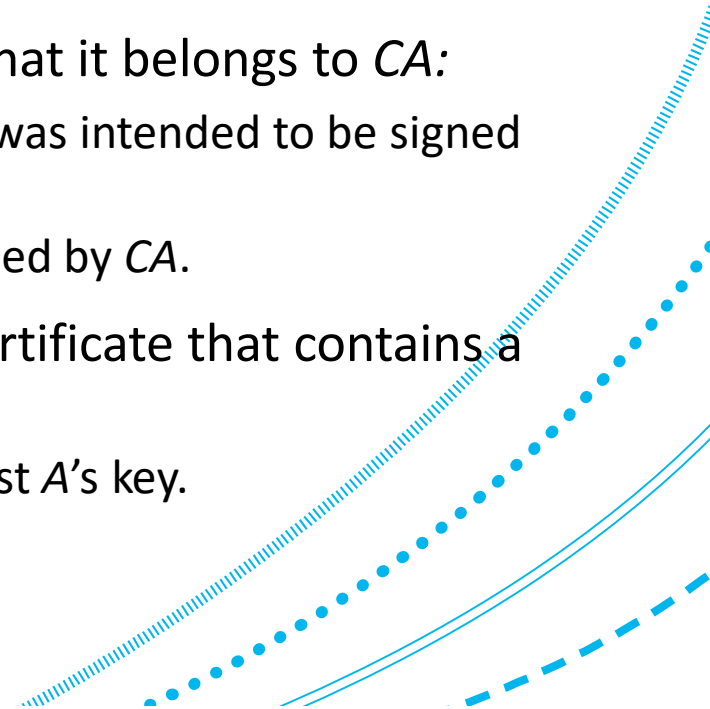
Certificates and proof of identity

1. Mere possession of a valid certificate does not identify the entity in possession as the entity named in the certificate.
2. Entity *A* could hold a certificate containing *B*'s name and public key.
3. Challenge-response protocols are often used for entity authentication instead.
 1. *B* might be issued a challenge, which requires a signature with its private key.
 2. Assuming that *B* has kept the private key private, this proves the identity of *B*.



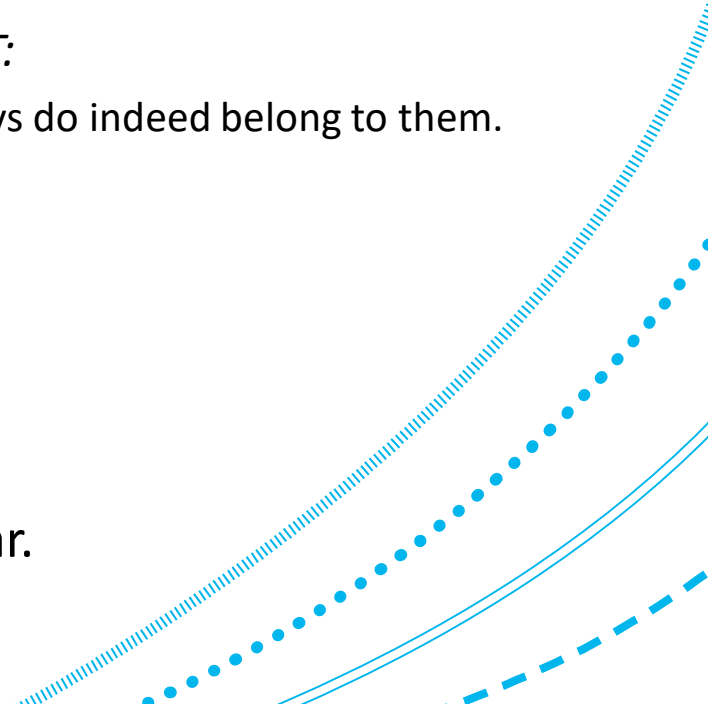
Certificate, key use and trust

- Suppose that there is a certificate, c , containing principal's name A , and public key v_A corresponding to private key s_A .
- The certificate will be signed by some other entity CA using some private key s_{CA} .
- Entity B might trust CA 's key, in the sense that it accepts that it belongs to CA :
 - It will therefore accept that anything signed with s_{CA} really was intended to be signed by CA .
 - In particular, it accepts certificate c was intended to be signed by CA .
- Furthermore, B might also trust that CA will only sign a certificate that contains a correct binding:
 - In this case, B will accept that v_A belongs to A , i.e. B will trust A 's key.
 - CA is a **certification authority (CA)** for B .



Application: using a key certification centre

1. If A and B both trust the same CA, T , to sign certificates binding their identities and public keys correctly, then they can use T as a **key certification centre**.
2. They use this as follows:
 1. A and B both have their own public-private key pairs.
 2. The public keys are put in separate certificates signed by T :
 1. Both A and B are then confident that each other's public keys do indeed belong to them.
 3. A encrypts the session key with B 's public key:
 1. So, A is confident that only B can decrypt.
 4. A signs the result with its own private key:
 1. So, B is confident that message came from A .
 5. Encrypted session key and signature can be sent.
3. Note that T does not get (symmetric) session keys in clear.



Certification authorities

1. A **certification authority (CA)** is there to provide digitally signed certificates that bind public key values to unique identities of entities.
2. CA is a form of *trusted third party (TTP)*.
3. This means that the CA's own public key must be trusted!
 1. Have replaced one trust problem with another.
 2. If attacker can get CA's private key, then B's can also be subverted.
 3. Since CA's sign for many entities this makes them juicy targets.
 4. There have been a number of such successful attacks.



Certifying certificates

1. But how do we know that the certificate authority's key really belongs to it? We just have the same problem all over again.
2. We could have it certified.
3. We end up with a chains of certificates and keys that we trust. These have to terminate somewhere.

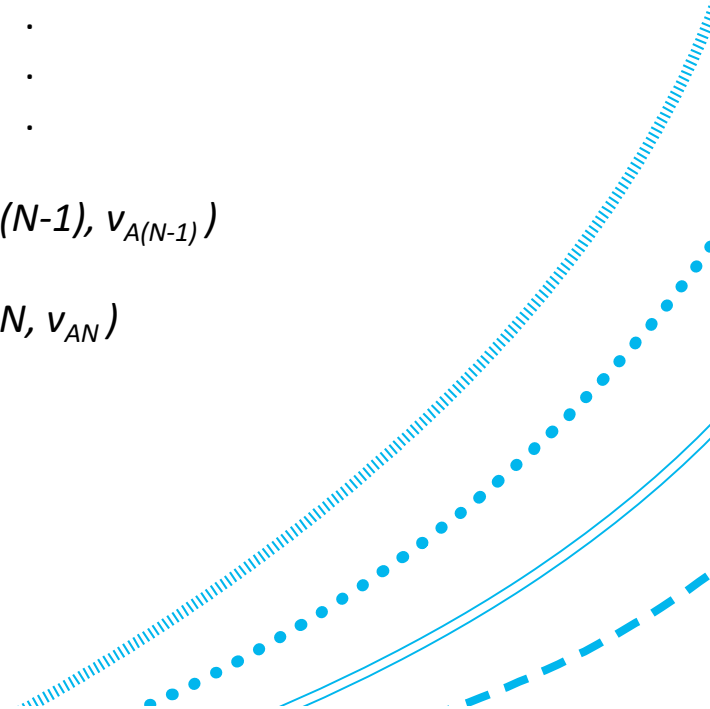
$Sig_{A0}(A1, v_{A1})$

$Sig_{A1}(A2, v_{A2})$

⋮

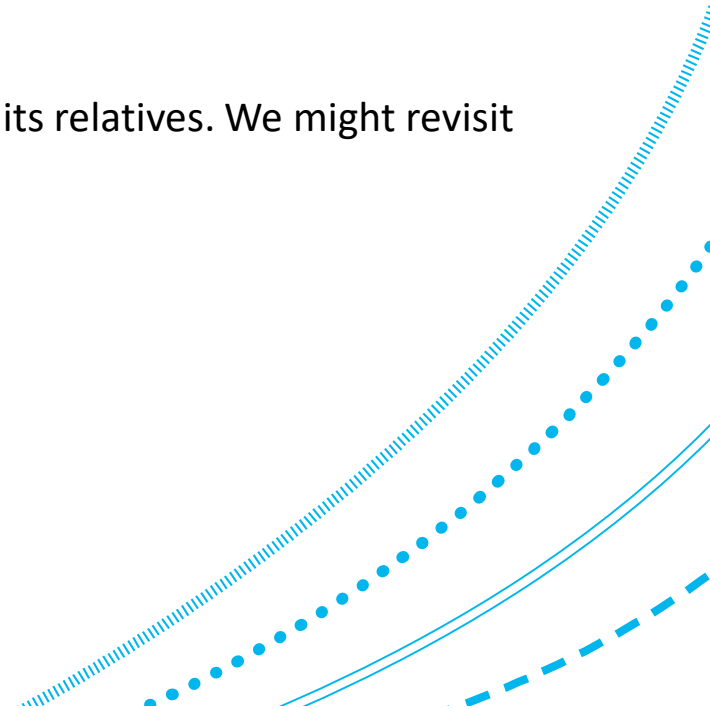
$Sig_{A(N-2)}(A(N-1), v_{A(N-1)})$

$Sig_{A(N-1)}(AN, v_{AN})$



Certifying certificates (cont'd)

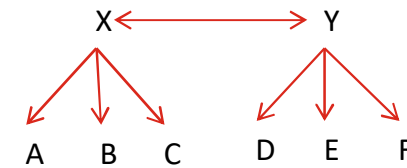
1. How do we decide to arrange larger systems of certificates and chains.
2. Two basic ways (currently):
 1. Centralised and hierarchical (e.g. PKIX / X.509 below).
 2. Decentralised and ad-hoc structure:
 1. This kind of thing arises with Pretty Good Privacy (PGP) and its relatives. We might revisit this later.
3. Different kinds of `trust' end up baked into the system.



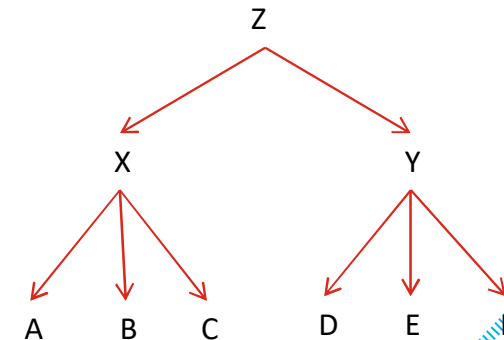
Arranging CAs

1. Suppose:
 1. A has a certificate from CA X.
 2. D has a certificate from CA Y.
 3. A wants assurance about authenticity of D's public key
2. Then A needs a copy of Y's public key.
3. The system needs one of:
 1. **Cross certification**
 1. X and Y issue certificates to each other
 2. **Certification hierarchy**
 1. X and Y both sit under another parent CA, say Z, that issues certificates to its children.

(1) Cross certification



(2) Certification hierarchy

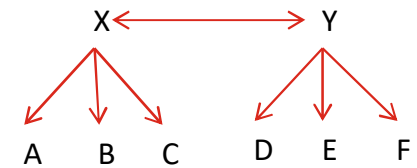


$M \rightarrow N$ means that M issues a certificate to N.

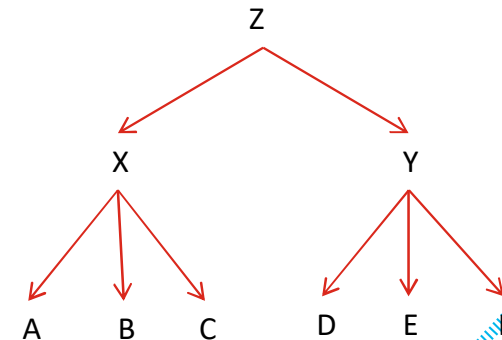
Using the CA arrangements

1. Suppose that *B* needs to trust *E*'s public key.
2. In situation (1):
 1. *B* needs to verify cert. of *Y* issued by *X*, and
 2. *B* needs to verify cert. of *E* issued by *Y*.
3. In situation (2):
 1. *B* needs to verify cert. of *Y* issued by *Z*, and
 2. *B* needs to verify cert. of *E* issued by *Y*.

(1) Cross certification

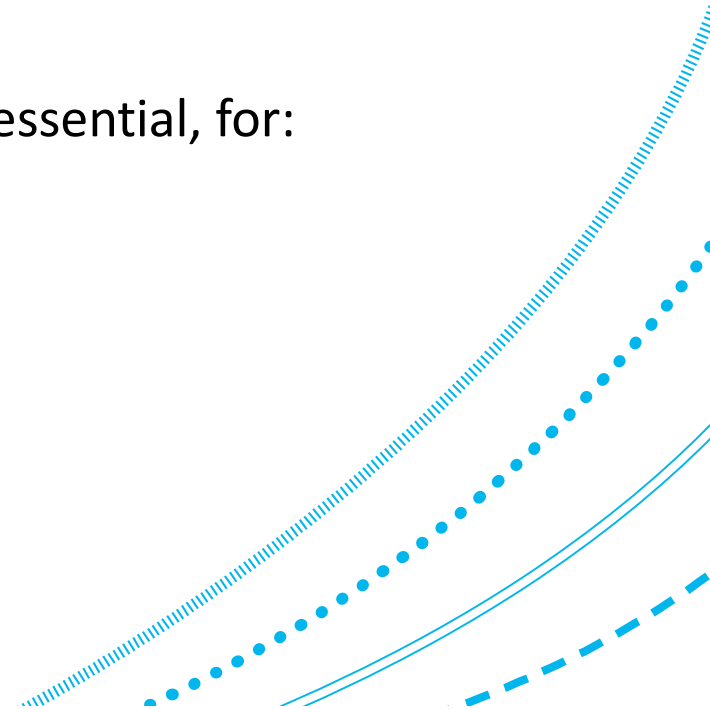


(2) Certification hierarchy



Public key infrastructure (PKI)

1. A **public key infrastructure** (PKI) is a system consisting of hardware, software, people, policies and procedures needed to create, manage, store, distribute, revoke and support use of public key certificates (PKCs).
2. They are widely used to underpin e-commerce.
3. They are currently critical, and sometimes argued to be essential, for:
 1. Secure email,
 2. Secure web-server access,
 3. Secure virtual private networks, and
 4. Other secure communications applications.

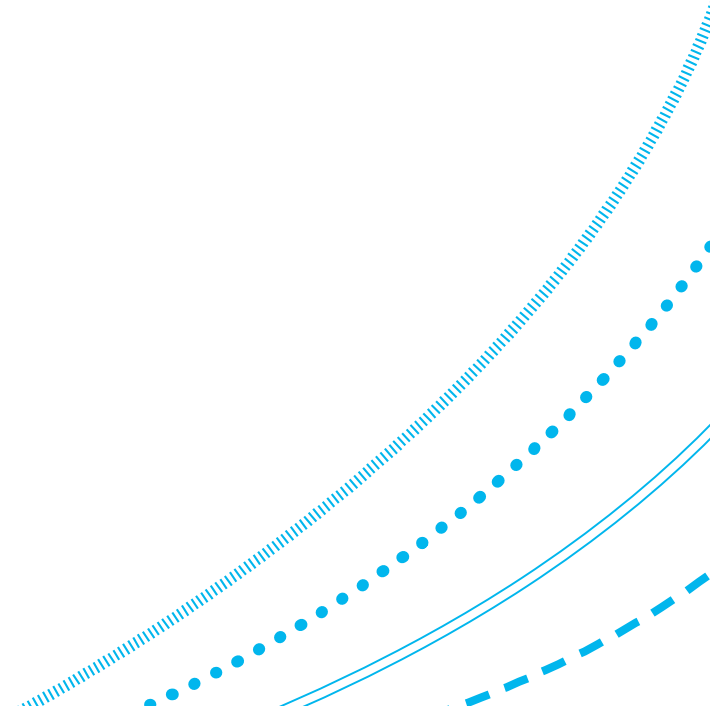


Need for PKI

1. Identity verification of central importance.
2. Cross verification requires CAs to use compatible technologies.
3. When should a CA be trusted?
4. CAs need to publish policy and practice statements containing clear information about their security.

Main players in a PKI system

1. Certificate owner :
 1. Applies for certificate.
2. CA:
 1. Issues certificate.
 1. This binds owner identity to owner public key value.
3. User ('relying party'):
 1. Uses the certificate for verification.



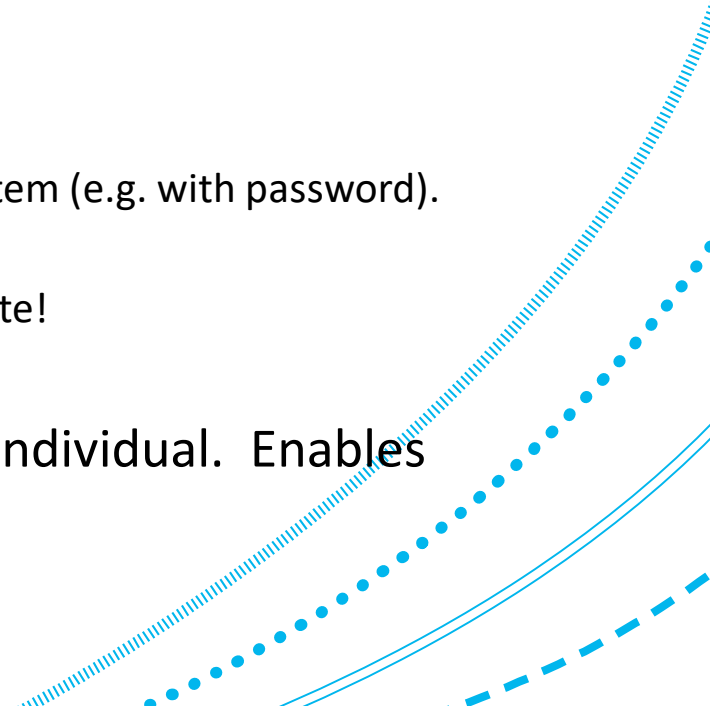
Validation authority

1. For a user, verifying the authenticity of a public key may involve verifying the signatures in a long chain of certificates:
 1. We saw small examples (a) and (b) above.
2. Possibly too expensive/time-consuming for users.
3. Instead, the task can be delegated to an entity to called a **validation authority (VA)**.
 1. So, the user relies on the VA.

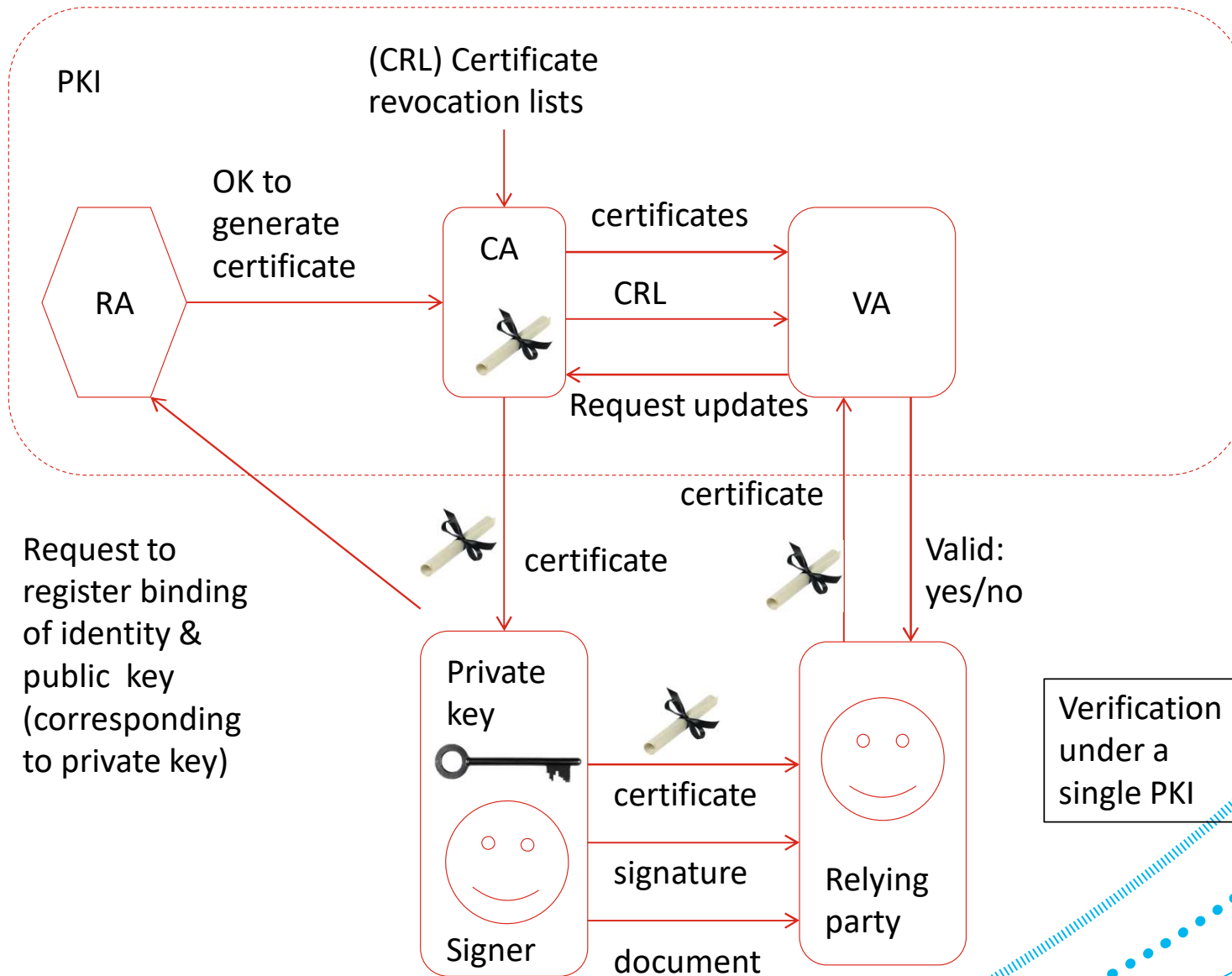


Registration authority

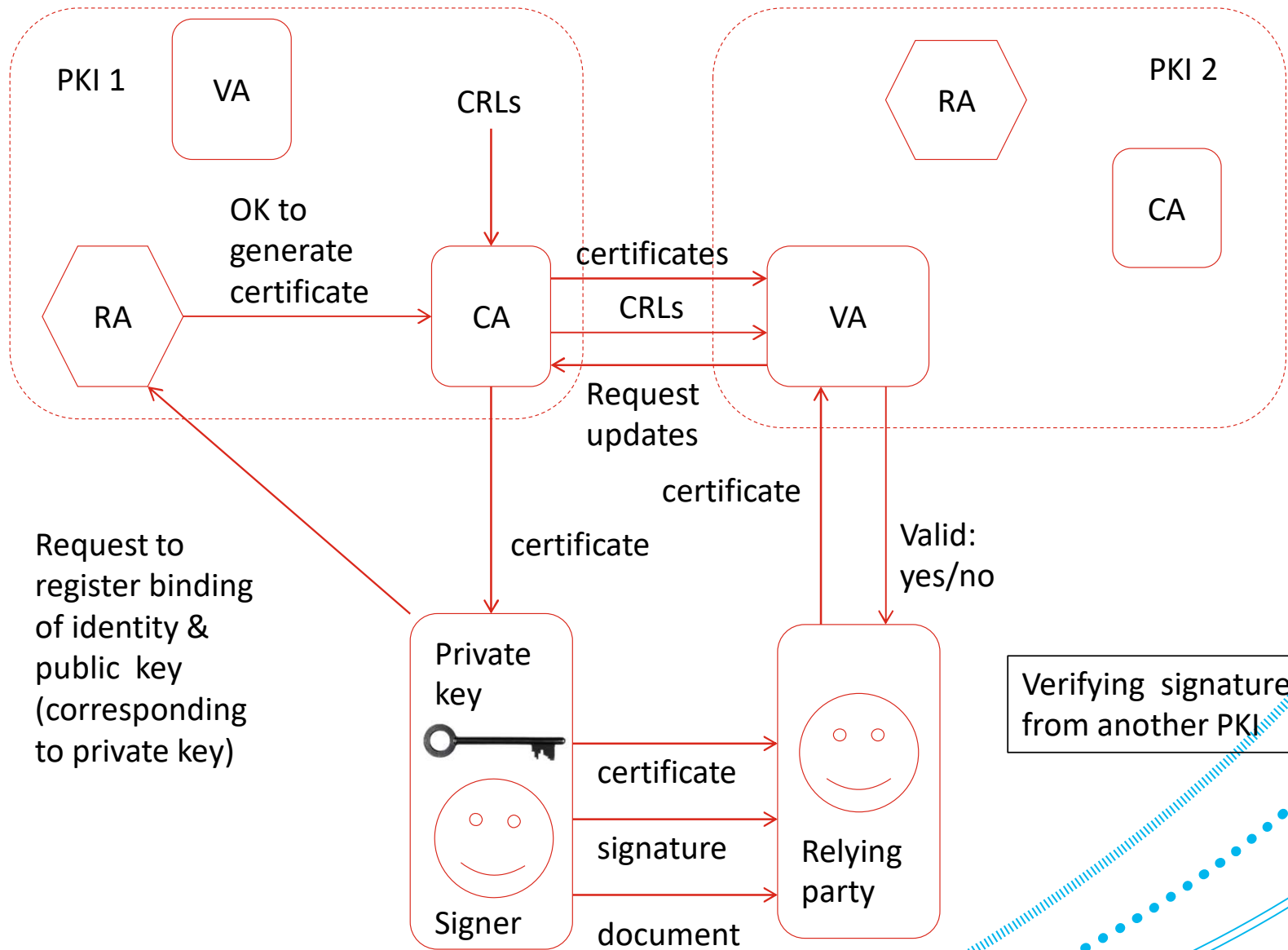
1. Sometimes the process of issuing certs is split up.
2. Putative owner, *O*, applies to a **Registration authority (RA)**:
 1. Verifies (somehow) the *O*'s identity.
 2. How can *O* prove their identity?
 3. Typically:
 1. Might be sufficient that the *O* can authenticate to some system (e.g. with password).
 1. Something they know.
 2. Might be that the *O* is required to produce another certificate!
 1. Something they have.
3. The RA and CA ensure that the key is bound to just that individual. Enables non-repudiation.



Verifier trusts
PKI



Verifier trusts PKI2,
but PKI2 also trusts
PKI1.



PKIX

X.509 Public key cert

Subject
Issuer
Serial Number
Value (public key)
Signature

X.509 Attribute cert

Holder
Issuer
Serial Number
Attributes
Signature

1. PKIX is the X.509 public key infrastructure for the internet.
2. This is constructed using standard certificate formats, and standard protocols concerning how they are managed and processed.
3. A **public-key certificate** (PKC) contains a subject's public key and some further information.
4. An **attribute certificate** (AC) contains a set of attributes for a subject (owner/holder). These are also known as **authorization certificates**. They bind the identity of the subject to the attributes: these are typically privileges.
5. Attribute certificates are issued by **attribute authorities**. A **privilege management infrastructure** (PMI) is a system of ACs, with their issuing attribute authorities, subjects, relying parties, and repositories.

PKIX entities

1. RA: registration authority
2. CA: certificate authority
3. CRL issuer: certificate revocation list issuer.
4. End entity: subject of certificate, or user of certificate.

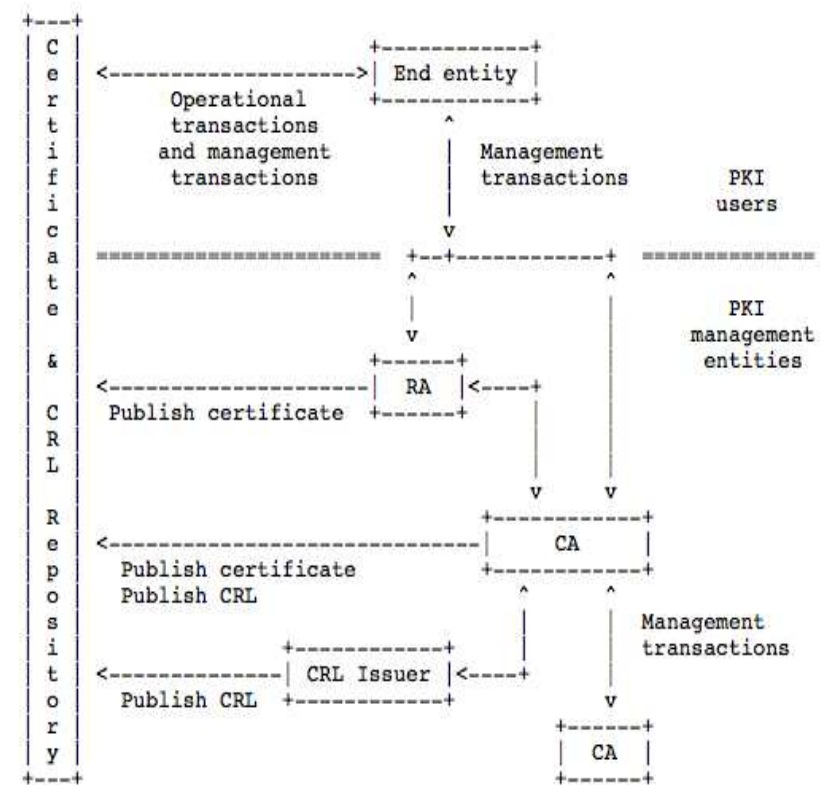
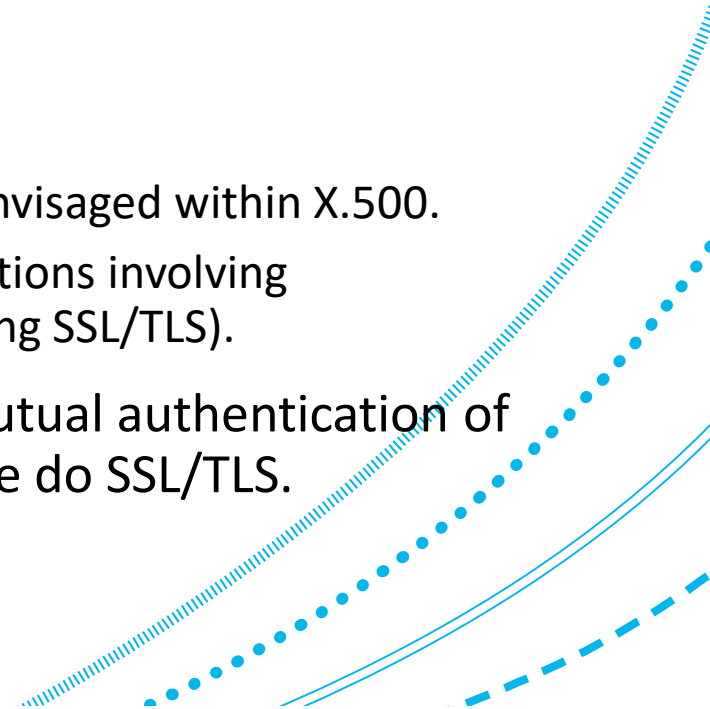


Figure 1. PKI Entities

X.509

1. X.509 (v3) is the most common PKI standard.
2. The X.500 Directory is a hierarchical tree structure able to contain names of devices, people, roles, etc. and corresponding attributes:
 1. Access control often tied to the identities in the directory.
3. X.509 is a later addition to this family of standards:
 1. Applications of X.509 are different than those originally envisaged within X.500.
 2. Applies to a broader access control situations (many situations involving cryptographic authentication and communication, including SSL/TLS).
4. X.509 can also be used for one-way authentication or mutual authentication of identities. We will discuss an application of this when we do SSL/TLS.



Structure of X.509 (v3) certificates

- **Version (v3)**
- **Serial Number:** issuer assigns integer (unique to it) for this certificate.
- **Signature Algorithm ID:** identifier for the alg. used to sign certificate.
- **Issuer name:** uniquely identifies issuer at any time, an X.500 name
- **Validity period**
 - **Not Before:**
 - **Not After:**
- **Subject Name:** Name for the subject to whom this cert refers. This certificate certifies the public key of the subject who holds the corresponding private key.
- **Subject Public Key Info:**
 - **Public Key Algorithm:**
 - **Subject Public Key:**
- **Issuer Unique Identifier:** (optional) Issuer names sometimes recycled
- **Subject Unique Identifier:** (optional) Subject names sometimes recycled
- **Extensions:** (optional) (v3 only)
 - ... ←
- **Certificate Signature Algorithm:** used for the digital signature below
- **Certificate Signature:** the signature of this certificate by the issuer.

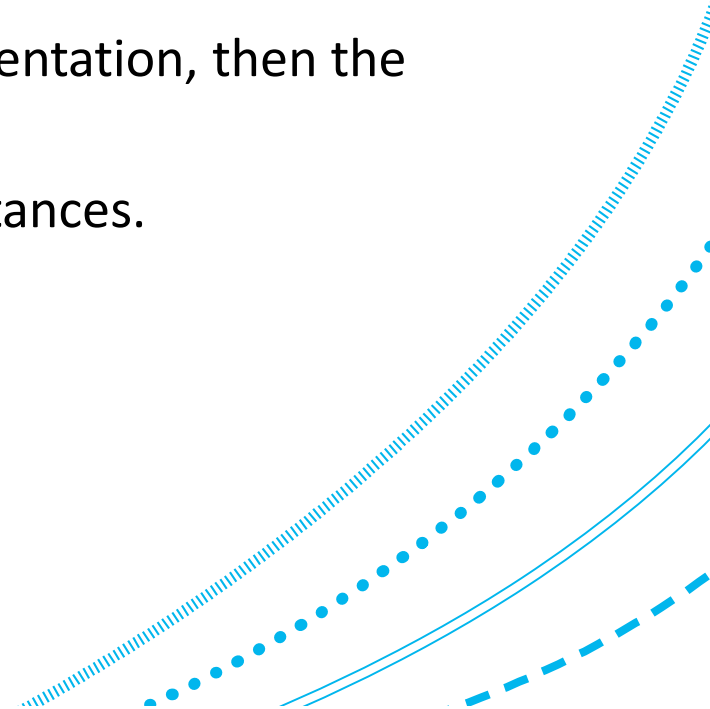
extensionID
critical: yes/no
extensionValue

X.509 extensions

1. **Key and policy information:** convey additional information about the subject and issuer keys, plus indicators of *certificate policy*.
 1. A **certificate policy** is a named set of rules that indicates the applicability of a certificate to a particular community or class of application with common security requirements.
2. **Certificate subject and issuer attributes:** can convey additional information and alternate names for subject and issuer. Used to give additional assurance to a user of the certificate.
3. **Certification Path Constraints:** These allow constraint specifications to be included in certificates issued for CAs, by other CAs - the issuer places the constraint. They include **basic constraints**, **name constraints** and **policy constraints**.

Critical extensions in X.509

1. Extensions can be marked as **critical**.
2. Critical extensions can be used to standardize policy with respect to the use of certificates.
3. If a critical extension cannot be processed by an implementation, then the certificate must be rejected.
4. Non-critical extensions may be ignored in some circumstances.



X.509 certificate: an example

```
dtp-3:Desktop mjcollin$ openssl x509 -inform der -in radius.abdn.ac.uk.cer -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            11:21:62:a6:1e:0b:c8:aa:ff:e8:9f:2f:89:b3:61:3b:c7:88
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=BE, O=GlobalSign nv-sa, CN=GlobalSign Organization Validation CA - G2
        Validity
            Not Before: Jul 11 15:40:02 2012 GMT
            Not After : Jul 11 15:31:51 2017 GMT
        Subject: C=GB, ST=Scotland, L=Aberdeen, O=University of Aberdeen, CN=radius.abdn.ac.uk
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
            Modulus (2048 bit):
                00:c0:06:61:02:61:56:98:28:f6:cd:22:39:da:8d:
                6c:0b:2f:32:c1:4a:4f:3d:dc:14:b0:0a:2b:bb:45:
                8b:e4:9c:29:2b:00:d6:af:16:e9:13:f9:79:0f:a3:
                c0:e5:f9:b1:f4:82:60:1b:7d:53:12:42:97:f3:f1:
                15:d8:27:db:4f:99:2c:78:f8:b7:30:59:a3:93:18:
                b4:8a:c1:57:97:b9:b1:87:7b:28:e5:b3:06:a5:db:
                2f:cf:fc:68:a8:e4:35:c2:e5:c9:db:0c:ae:a5:ca:
                3b:b8:57:3a:a2:9d:76:e4:d4:61:e9:41:52:43:a6:
                75:89:02:98:93:b2:73:4b:8f:b0:a0:4f:19:96:39:
                58:90:49:b5:11:15:5d:d1:5e:27:49:f5:38:b6:9f:
                ef:73:a6:f9:17:a0:f8:99:8c:b3:97:85:e4:c3:aa:
                75:d8:fa:35:e5:78:bb:d8:0b:aa:34:94:f0:02:45:
                73:f9:5e:38:23:cd:bc:7c:6a:1b:66:3e:3f:7e:64:
                3d:29:58:ca:0a:e9:f6:be:5e:88:91:ad:3a:21:f1:
                47:27:a2:ad:aa:fd:49:79:78:21:73:a6:fb:51:60:
                18:12:6a:91:ad:a6:44:b0:f4:35:5b:22:7e:ad:7f:
                c0:41:a7:61:45:0e:70:87:ee:27:f9:44:26:6d:22:
                5a:43
            Exponent: 65537 (0x10001)
```

X.509 certification: an example (cont'd)

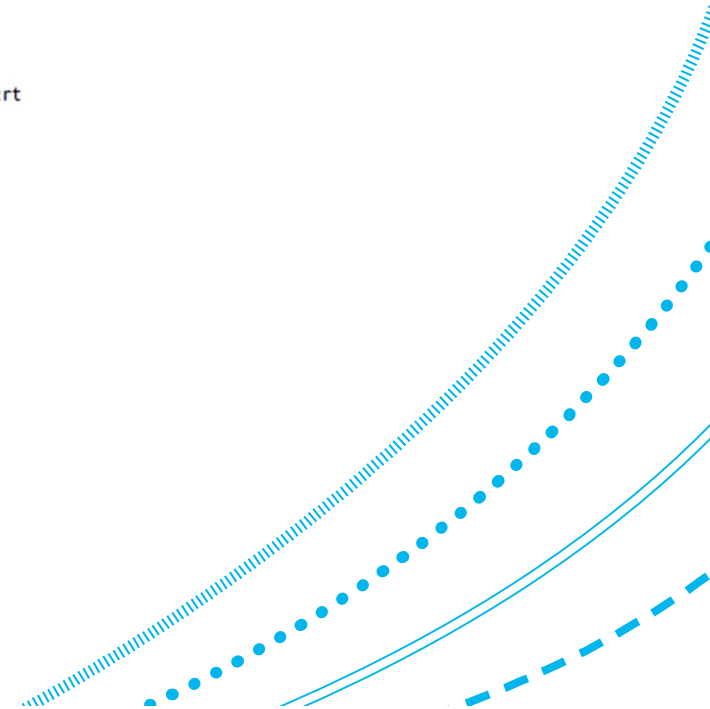
```
X509v3 extensions:
  X509v3 Key Usage: critical
    Digital Signature, Key Encipherment
  X509v3 Certificate Policies:
    Policy: 1.3.6.1.4.1.4146.1.20
    CPS: https://www.globalsign.com/repository/

  X509v3 Subject Alternative Name:
    DNS:radius.abdn.ac.uk
  X509v3 Basic Constraints:
    CA:FALSE
  X509v3 Extended Key Usage:
    TLS Web Server Authentication, TLS Web Client Authentication
  X509v3 CRL Distribution Points:
    URI:http://crl.globalsign.com/globalsignvalg2.crl

  Authority Information Access:
    CA Issuers - URI:http://secure.globalsign.com/cacert/globalsignvalg2.crt
    OCSP - URI:http://ocsp2.globalsign.com/globalsignvalg2

  X509v3 Subject Key Identifier:
    F7:8D:DE:93:AF:A8:EF:FE:4F:DB:C4:93:2E:8D:20:29:A5:C0:6E:8B
  X509v3 Authority Key Identifier:
    keyid:5D:46:B2:8D:C4:4B:74:1C:BB:ED:F5:73:B6:3A:B7:38:8F:75:9E:7E
```

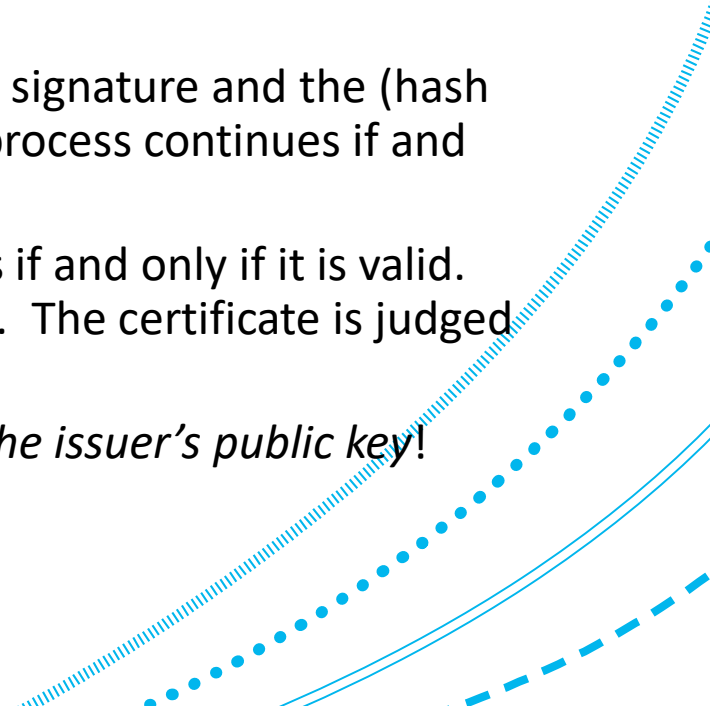
```
Signature Algorithm: sha1WithRSAEncryption
83:c4:27:e4:c3:1f:4a:f6:d0:68:97:c2:82:9a:ec:0f:d2:a9:
d5:cc:cb:74:5e:ff:df:7f:5a:4c:fc:b2:8a:29:07:1f:7e:1a:
9b:d5:fd:5f:b9:54:2e:ce:1e:bc:2c:91:52:2c:79:0e:99:90:
bf:51:07:e0:5a:59:a9:cd:90:87:45:68:28:5d:8c:66:7f:4a:
46:1f:1c:23:ed:ad:40:cd:d9:de:0a:83:0d:17:3c:b6:fa:f7:
b0:be:14:ec:cd:70:b0:60:d3:b5:df:ca:c3:e2:91:22:55:7a:
bf:70:26:6e:a3:0f:5f:f2:b8:a5:7c:95:3c:b6:0b:7e:eb:7d:
98:97:e0:1a:9b:b9:a8:aa:c0:1e:f9:6b:63:ec:10:75:36:9a:
e9:ad:0f:54:76:21:d4:a4:be:c7:d0:a2:8a:fc:b7:e7:b8:d3:
10:4c:02:b8:e2:73:05:2b:80:bb:04:97:fc:17:b0:7d:11:fa:
17:59:a7:30:b4:47:4b:a5:22:06:db:c3:04:dd:1a:52:ce:5c:
65:5e:42:16:9d:48:6f:30:14:36:00:1d:ff:ea:9b:20:d1:ee:
aa:f7:48:74:47:5d:8f:c3:64:13:fb:c4:7a:57:61:e5:77:45:
c9:00:61:60:30:35:ec:91:18:cf:ce:ff:1e:2a:08:1b:5f:63:
d9:bb:77:94
```



Basic validation story

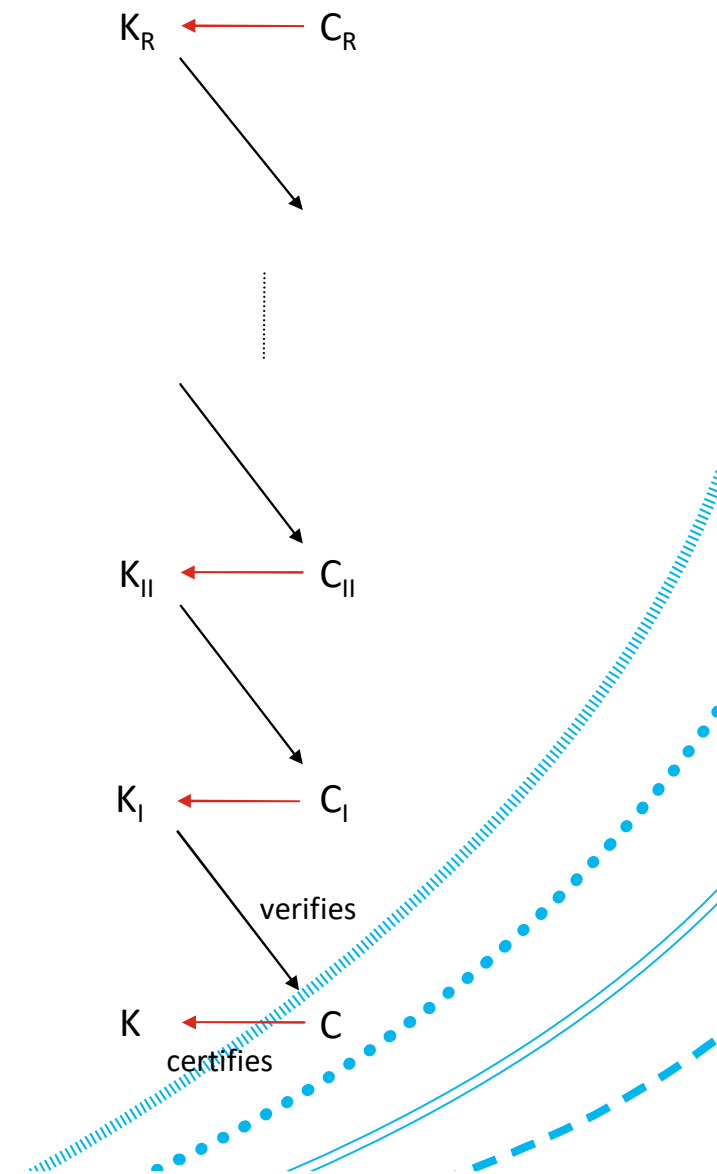
If a validation authority is doing this, replace the word “user” with “VA”. Also, “user” usually means “agent of user”.

1. Assuming the certificate is not revoked, the following happens:
 1. User obtains the issuer’s public key for the particular algorithm:
 1. This is not on this certificate (except in the self-signing case below)
 2. We must get it from elsewhere (we’ll revisit this below).
 2. User applies the appropriate verification algorithm to the signature and the (hash of the) foregoing fields in the certificate. The validation process continues if and only if the signature is verified as correct.
 3. User then checks the period of validity. Process proceeds if and only if it is valid. Finally, the *constraints in the extensions* must be checked. The certificate is judged to be valid if and only if these checks are passed.
 4. *Note that at step 2 we are relying on the authenticity of the issuer’s public key!*



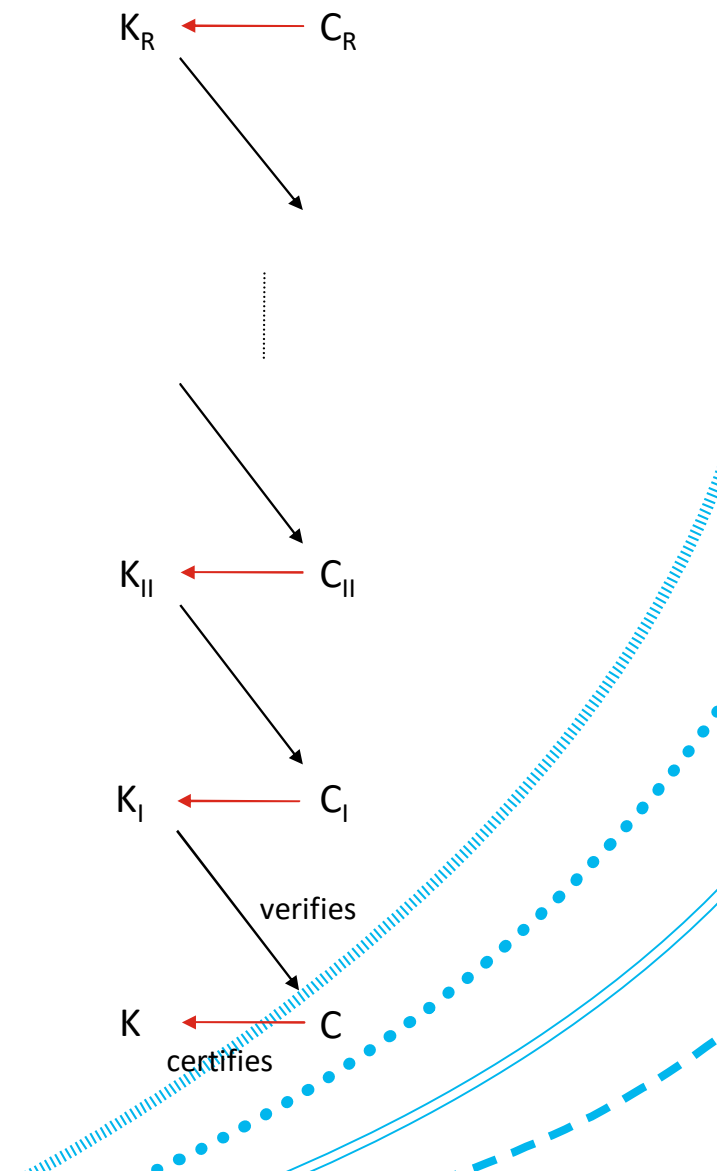
Certificate chains

1. On our 'basic validation' slide above, we needed the issuer's public key K_I in order to perform the validation of the certificate, C (which certifies some K).
2. We need to be sure that K_I is authentic. Often, this assurance is provided by another certificate C_I .
3. C_I will have been provided by the issuer-issuer, II , who has a public key K_{II} for which there is a certificate C_{II} . We need to verify C_{II} , and so on.
4. We can thus require a **chain of certificates** (and keys) as shown in the picture.



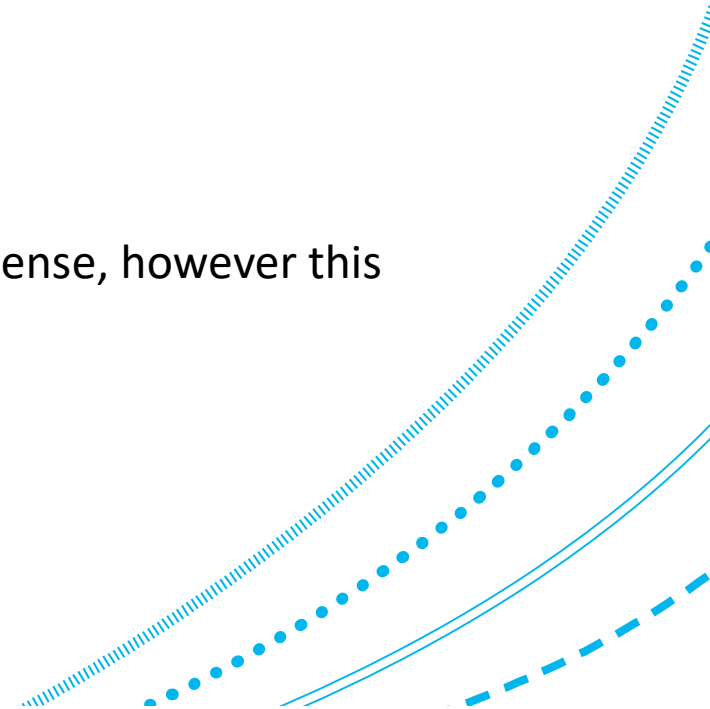
Certificate chains (cont'd)

1. If all is well, then this should end with a **root verification key**, K_R . There is sometimes a root certificate, C_R , as well.
 1. The authenticity of this is not assured by another certificate. It is trusted, i.e., assumed to be genuine.
 2. Typically, these are stored somewhere locally and securely, by the entity that wishes to verify C .
 3. Everyday example is that browsers and operating systems are shipped with a small number of root-certificates and root-keys.



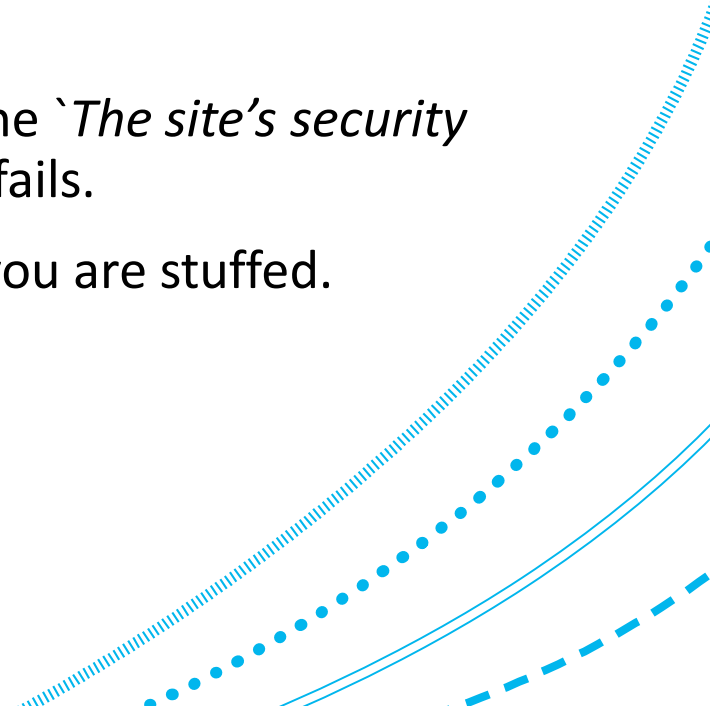
Root keys and certificates

1. The root key K_R (e.g. in your browser or OS) is the public key half of a public-private key pair.
2. The private half of the pair, say K_R^S , is also known as the Root Key. It is held by a Certificate Authority under tight security.
3. The public root key is often **self-signed**.
 1. What this means is that C_R has been signed with K_R^S .
 2. Hence, K_R is the only key that can 'verify' C_R , in the usual sense, however this doesn't mean much.



Validation revisited

1. Checking a certificate thus involves proceeding recursively back up the unique chain (from a root) that leads to it.
2. The process terminates either when it fails, or when you validate a certificate that was signed by a root certificate.
3. In a browser using certificates as part of https, you get the *'The site's security certificate is not trusted'* (or similar) message when this fails.
4. If an attacker can insert a new root pair K_R and C_R , then you are stuffed.



PKIX certification path processing

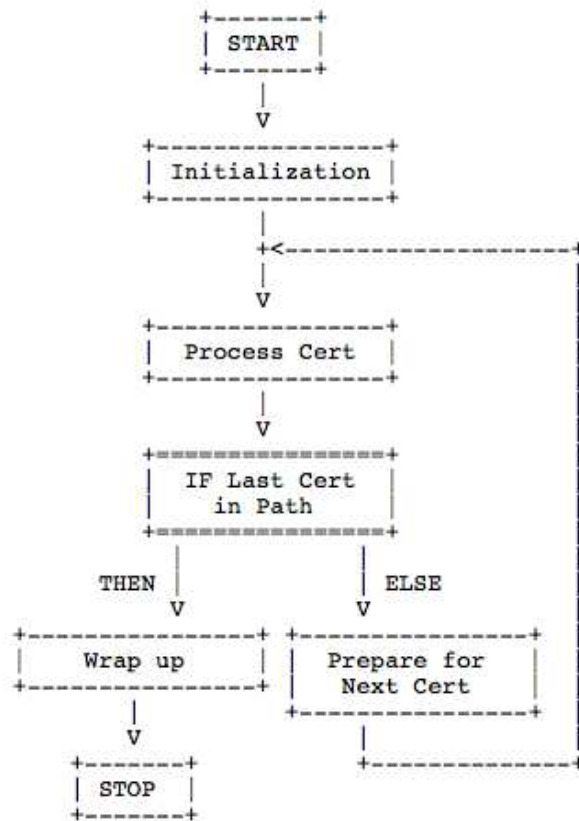
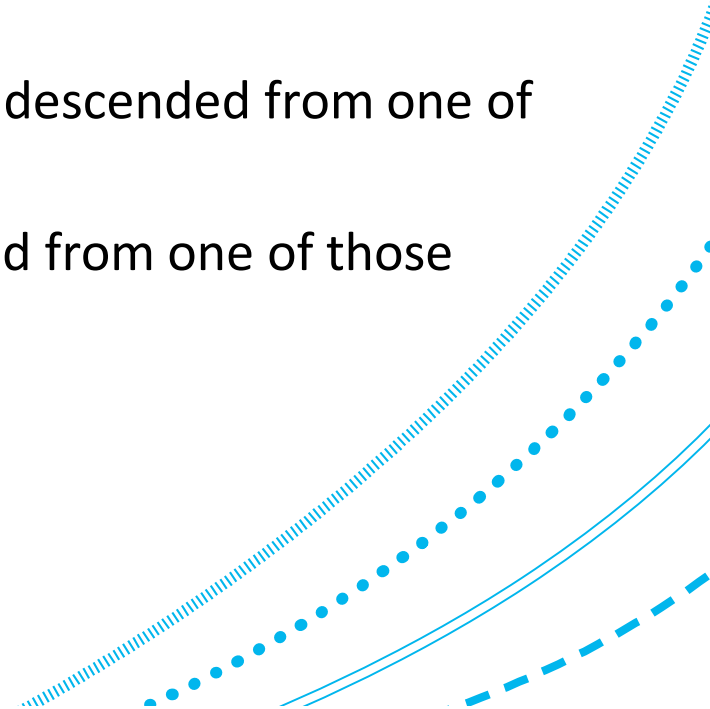


Figure 2. Certification Path Processing Flowchart

1. This is a rudimentary description from [RFC5280](#).
2. The real description of all the details (in natural language) is very complicated.
3. This causes implementation bugs.

Generating and getting certificates

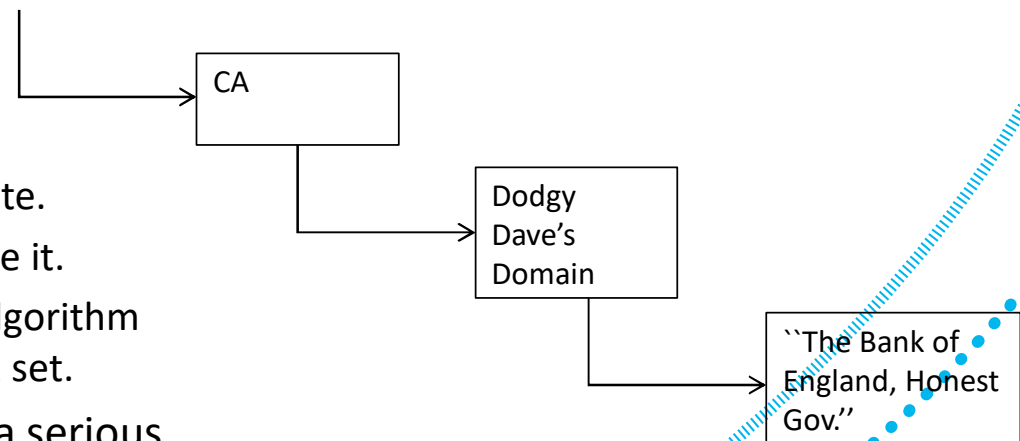
1. It is easy to generate your own root certificate:
 1. E.g., You can easily set up your own CA using openssl.
 2. You'd have to pay a lot of money (and demonstrate your trustworthiness) to have it embedded in a major browser distribution.
2. You can easily be issued with a certificate by a CA that is descended from one of the standard root CAs.
3. It is much harder to be certified *as a* CA that is descended from one of those standard roots.



The missing link: checking constraints

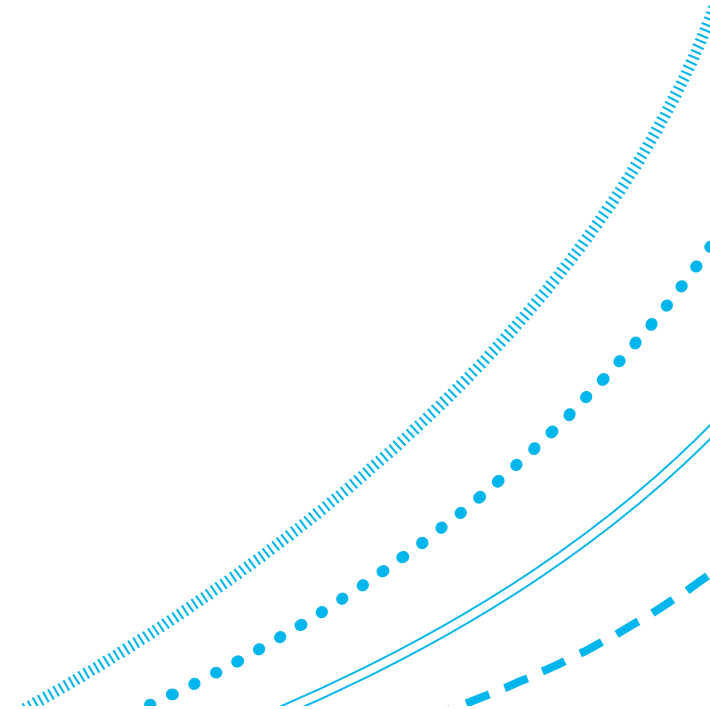
1. On our basic validation slide, I mentioned checking of constraints.
 1. It is perfectly possible for nefarious characters to get hold of a certificate.
 2. What is to stop someone dodgy with a genuine certificate from issuing a certificate that spoofs something genuine?

- Dodgy Dave should not have CA:
 - TRUE set in the critical extensions for his certificate.
 - That was issued by his issuer, so he cannot change it.
 - The implementation of the certificate checking algorithm should check that all of the issuers have CA:TRUE set.
- The failure to check such constraints resulted in a serious exploit of SSL.



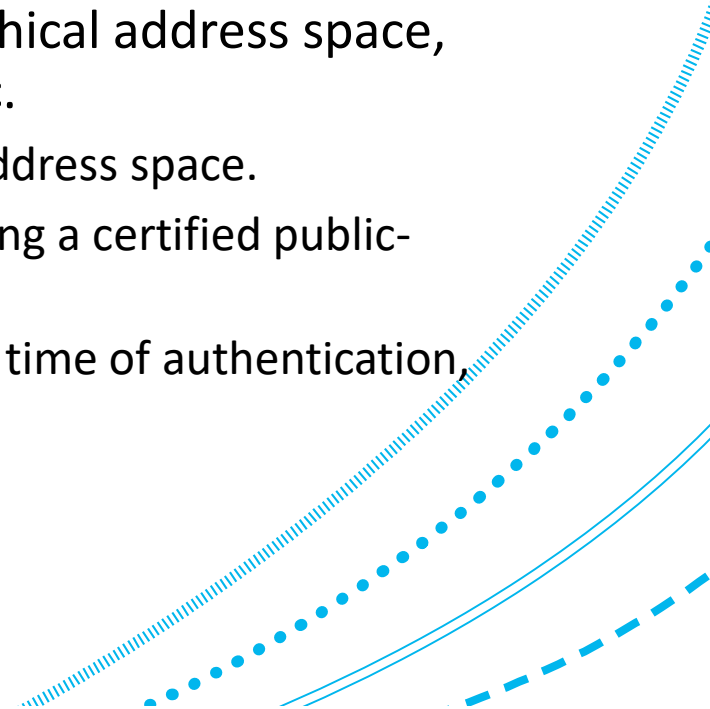
Expiry dates

1. What should we do when the expiry date of one of the certificates has passed?
2. Two ways commonly discussed:
 1. **Shell Model**
 2. **Chain model**



Shell model

1. Motivating idea: A CA should only issue certificates that expire before its own.
2. Mode of operation: If any certificate in the chain is not between its expiry dates, then we reject it and hence the whole chain below it.
3. Typical application: Suitable for subjects within a hierarchical address space, where the purpose is authenticating the source of traffic.
 1. The traffic traverses links (edges) between nodes in the address space.
 2. Each traversal of a single edge requires authentication using a certified public-private key pair.
 3. Confirm that all links to the address space are valid at the time of authentication, and only at that time.



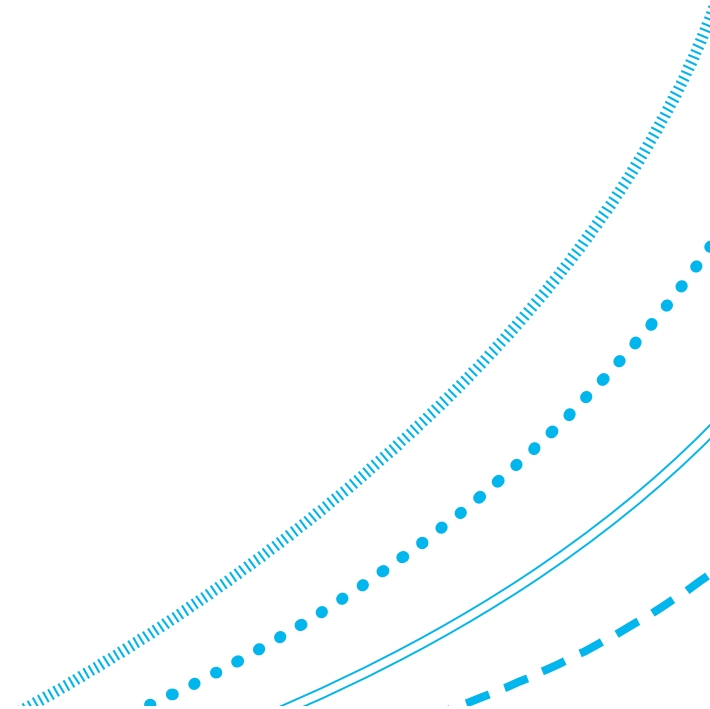
Chain model

1. Motivating idea: The issuer's certificate only had to be valid at the time that the certificate was issued.
2. Mode of operation: Certificates may remain valid even after a certificate above them in the chain has expired or has been revoked.
3. Typical Application: Authenticating documents.
 1. A digital document has been signed with a public-private key pair that has a certificate. If the certificate was valid at the time the document was signed, we should accept it.
 2. Analogy: Certificate of student status should remain valid, even if the registry officer that signed it later leaves.



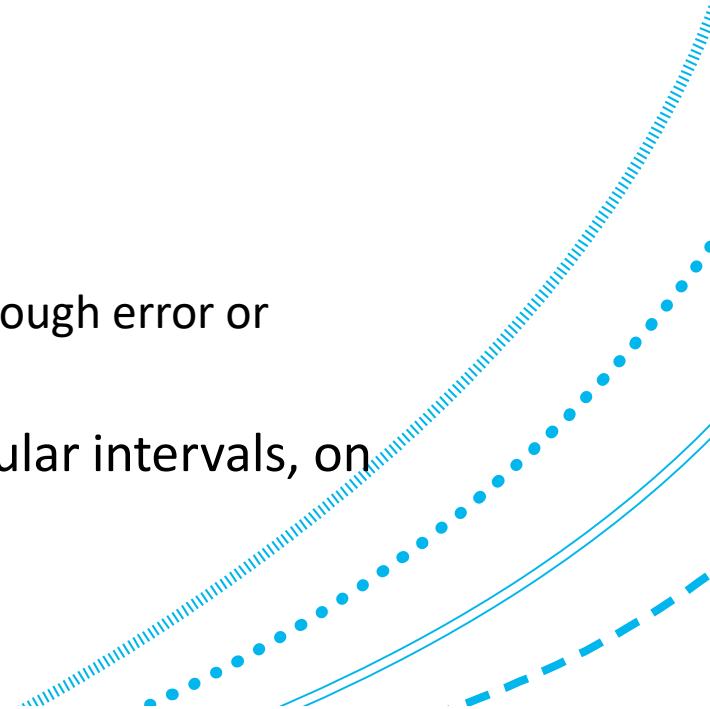
Chain model (cont'd)

4. Additional requirement: A secure and reliable time-stamping authority as a trusted third party.
 4. In the example above, we must know that the signature was made during the period of validity.



Revocation

1. A CA may wish to **revoke** a certificate that it has issued to a user (e.g., I may wish to revoke certificate C , which was being used to guarantee key K , in our chain example above).
2. For example, if:
 1. The user's private key (matching K) is compromised, or
 2. Another fact that the vouches for is no longer valid, or
 3. The user is no longer certified, or
 4. The certificate should never have been issued but was through error or compromise.
3. **Certificate Revocation Lists (CRLs)** are distributed at regular intervals, on demand, or sometimes queried on-line.



Problems with digital certificates

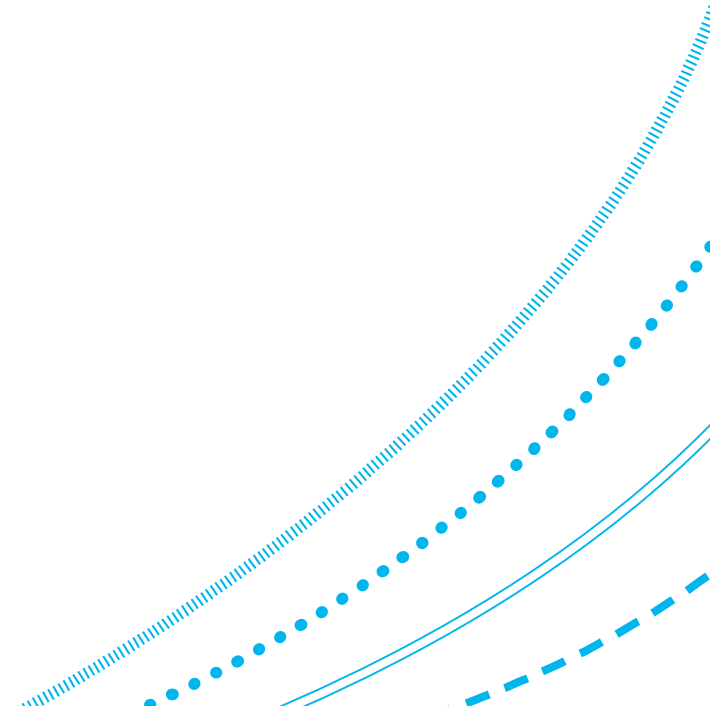
1. Revocation:

1. Examples:
 1. Company issues certificate to employee, who then leaves.
 2. Keyholder knows that their private key is compromised.
2. Certificate (together with private key) gives authentication capability.
3. CA needs to revoke certificate.
4. Difficult since certificates often widely distributed.
5. CA may issue a **certification revocation list (CRL)**
6. Known to be difficult to manage these and have various problems.

Problems with digital certificates (cont'd)

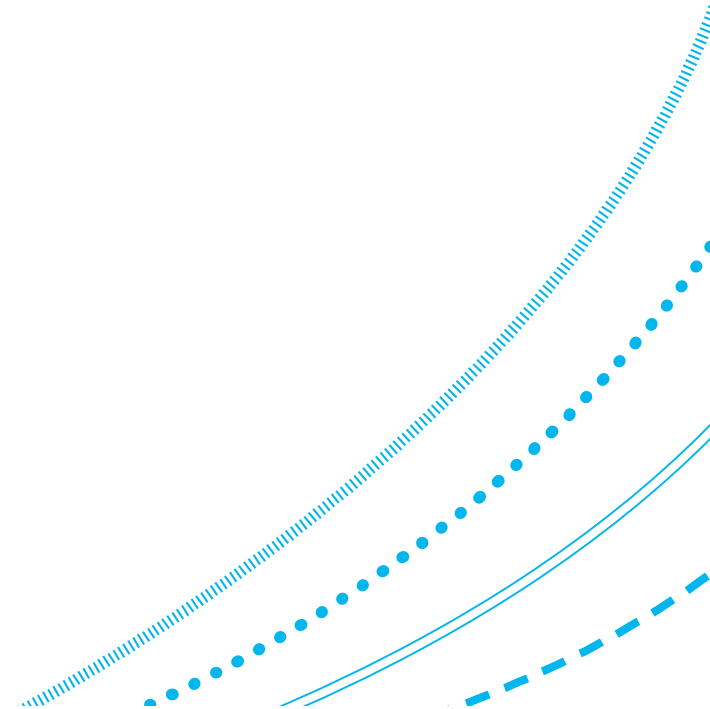
2. Liability:

1. Users rely on certificates from CA.
2. Certificate might be erroneous.
 1. Public key value listed may not belong to the listed owner.
3. If this is the case, who is liable?
 1. Owner (of public key)?
 2. User (person displaying it)?
 3. CA?
4. Not clear.



Problems with digital certificates and PKI

1. Complicated specifications lead to implementation problems (such as with the critical CA extension example above).
2. Some distributed systems engineers say that it is impossible to correctly implement a PKI.



Association of keys and access rights

1. X.509 and PKIX are **name-centric** PKIs:
 1. To associate a key with access rights, you have to know the subject to which a key belongs.
 2. Authorization attributes are linked to a cryptographic key via a common name in a PKC and an AC.
2. An alternative is the **simple public-key infrastructure** (SPKI), a key-centric PKI.
 1. SPKIs certificates bind keys directly to attributes (access rights).

