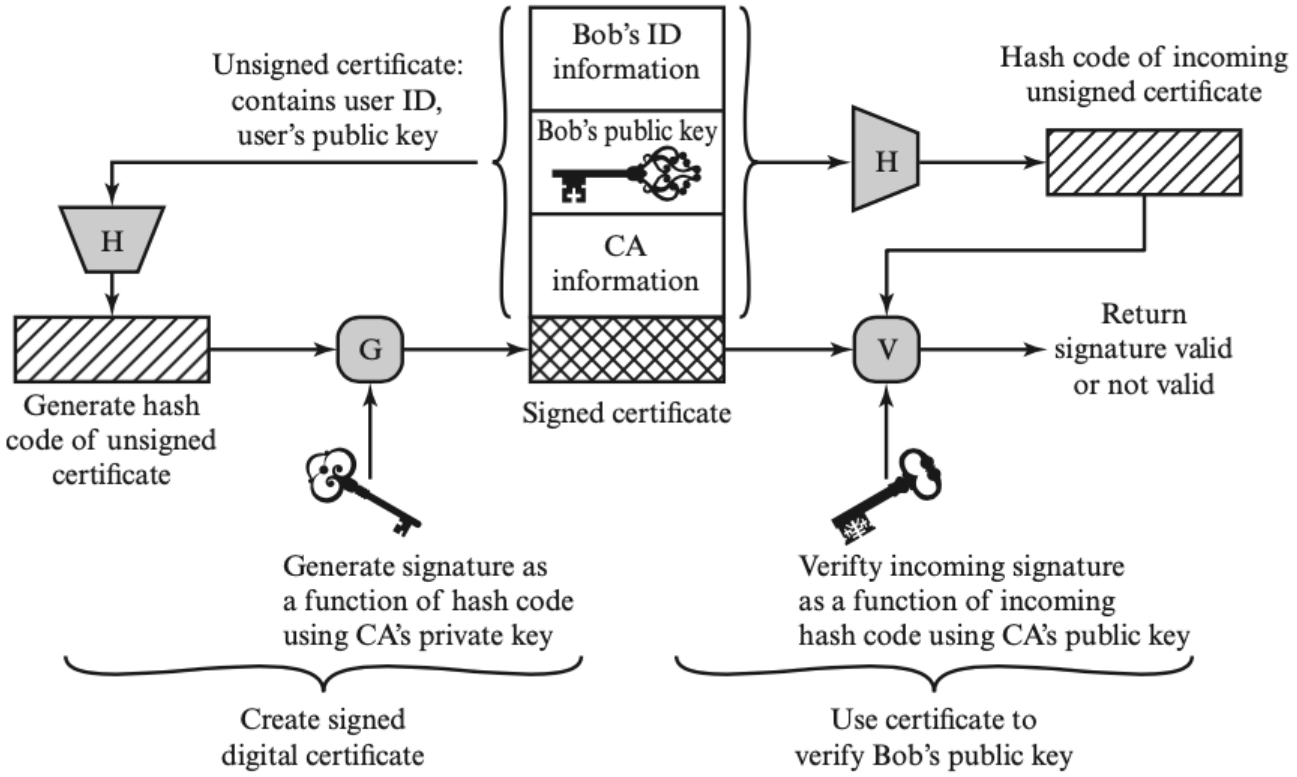
Certificates

Certificates

- **Certificate**: A signed endorsement of someone's public key
  - A certificate contains at least two things: The **identity** of the person, and the **key**
- Abbreviated notation
  - Signing with a private key *SK*: {"Message"}$_{SK^{-1}}$
    - Recall: A signed message must contain the message along with the signature; you can't send the signature by itself!
- Scenario: Alice wants Bob's public key. Alice trusts Charlie (PK_C, SK_C)
  - Charlie is our trust anchor
- If we trust PK_C, a certificate we would trust is {"Bob's public key is PK_B"}$_{SK_C^{-1}}$

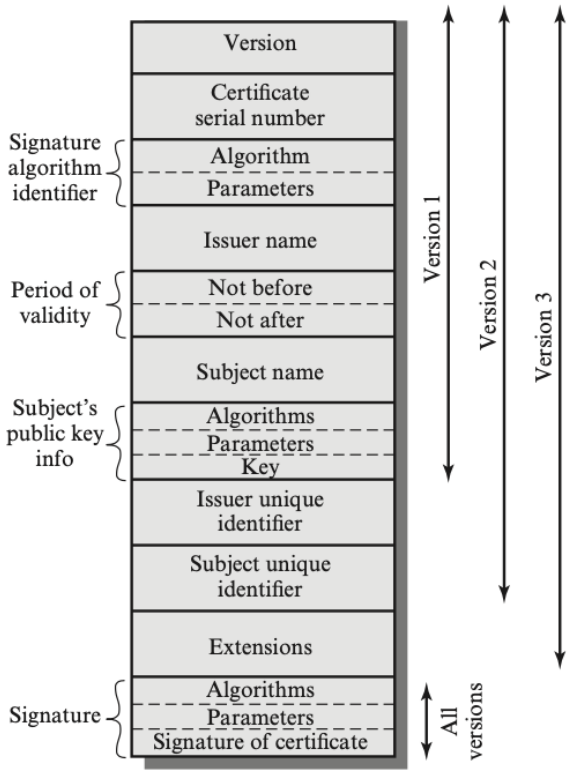
How do we use public-key certificate?



X.509 Certificates

- Certificate serial # - SN
- Period validity - T^A
- Subject's public key info - Ap
- Signature signed by CA's private key
- Math notation:

$$CA \ll A \gg = CA\{V, SN, AI, CA, UCA, A, UA, Ap, T^A\}$$



readings

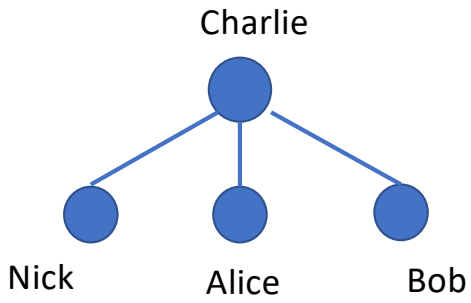
- Barnes, R.; Hoffman-Andrews, J.; McCarney, D.; Kasten, J. (March 2019). *Automatic Certificate Management Environment (ACME) RFC 8555*. IETF
- Internet Security Research Group, “Annual Report”, https://www.abetterinternet.org/annual-reports/
- Minkyu Kim, “A Survey of Kerberos V and Public-Key Kerberos Security”, https://www.cse.wustl.edu/~jain/cse571-09/ftp/kerb5/index.html

Obtaining a User Certificate

The Trusted Directory

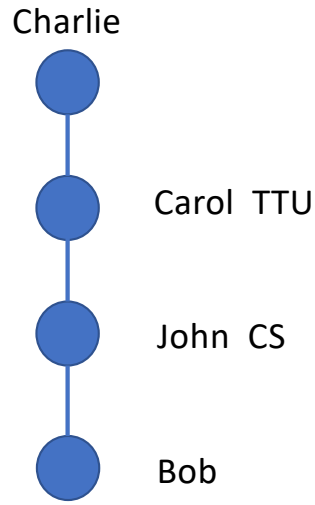
- Idea: Make a central, trusted directory (TD) from where you can fetch anybody's public key
  - The TD has a public/private keypair PK_{TD}, SK_{TD}
  - The directory publishes PK_{TD} so that everyone knows it (baked into computers, phones, OS, etc.)
  - When you request Bob's public key, the directory sends a certificate for Bob's public key
    - {"Bob's public key is PK_B"}$_{SK_{TD}^{-1}}$
  - If you trust the directory, then now you trust every public key from the directory
- What do we have to trust? (assumption)
  - We have received TD's key correctly
  - TD won't sign a key without verifying the identity of the owner

The Trusted Directory



- Let's say that Charlie (an authority) runs the TD
  - We want Nick's public key: Ask Charlie
  - We want Alice's public key: Ask Charlie
  - We want Bob's public key: Ask Charlie
  - Charlie has better things to do (like making sure his private key isn't stolen)!
- Problems?
  - Scalability: One directory won't have enough compute power to serve the entire world
  - Single point of failure:
    - If the directory fails, your application might become unavailable
    - If the directory is compromised, you can't trust anyone
    - If the directory is compromised, it is difficult to recover

Certificate Authorities



- Addressing scalability: Hierarchical trust
  - The roots of trust may **delegate** trust and signing power to other authorities
    - {"Carol Christ's public key is PK_{CC}, and I trust her to sign for TTU"}$_{SKc^{-1}}$
    - {"John Canny's public key is PK_{JC}, and I trust him to sign for the CS department"}$_{SKCC^{-1}}$
    - {"Bob's public key is PK_{Bob} (but I don't trust him to sign for anyone else)"}$_{SKJC^{-1}}$
  - Charlie is still the root of trust (**root certificate authority**, or **root CA**)
  - CC and JC receive delegated trust (**intermediate CAs**)
  - Bob's identity can be trusted
- When Alice wants to request Bob's PK, she can obtain this list of certificates (certificate chain) from any untrusted service, and does not need to contact the respective parties because the signatures are unforgeable
- Addressing scalability: Multiple trust anchors
  - There are ~150 root CAs who are implicitly trusted by most devices
  - Public keys are hard-coded into operating systems and devices

Review class & quiz 4

- Friday (Nov. 15) in class
- DHKE & Kerberos

- Thank you!