

Introduction to Computer Security Lecture Slides

© 2024 by [Mihai Chiroiu](#) & [Florin Stancu](#)

is licensed under [Attribution-NonCommercial-ShareAlike 4.0
International](#)

Public Key Infrastructure

Contents

- Public-key distribution problem
- Certificates
- Validation Types
- Case studies
 - TLS, SSH, PGP/GPG
 - OTR instant messaging services
 - Email security, DNS-based PKI etc.

Problem

- Digital signatures (also see: public key encryption):

`signature = Sign(message, priv_A)`

`valid = Verify(signature, pub_A)`

- Public key must be given to all interested parties...
- How? MitM may alter public keys => need integrity guarantees!
 - Transfer it over a secure channel?
 - Use one (or many) common **trusted party**?
 - Public announcement / ~~YOLO~~ TOFU

Solutions

- Public Key Infrastructure
 - a central authority that manages trust
 - trust is built-in (installed together with the OS)
 - requires a mechanism for verifying trust by the central authority
- Web of Trust (PGP)
 - relies on peer-to-peer (decentralized) trust transfer
 - requires a mechanism for “manual” key transfer
 - based on the transitive relationship of trust

Public Key Infrastructure

- Idea: store public keys in **public trusted** repositories
 - Operated by **trusted** authorities!
 - Must do: PK owner verification, costs / inconvenience?
- Trusted channel with authority, database integrity?
 - Digital signature of each “**public key entry**” with authority’s key!
- Multi-level hierarchies?
 - *Don’t put all eggs in the same basket!*
 - Intermediate authorities signed by higher-level authorities
 - Turtles all way down => trust anchor

Certificates

- Proof you have/did something of relevance 🤔
- Attributes:
 - Your identity
 - What does it prove
 - The issuing authority
 - Entitlements
 - Expiration date
 - ID / Registration number
 - custom metadata etc.
- Standard for certificates: X.509



X.500 standards

- X.500 – ITU specifications for Directory Services (e.g., LDAP)

- Identifying an entity:

- **DN** (Distinguished Name)

- **DN** fields:

- C: Country
 - O: Organisation
 - OU: Organisational Unit
 - DC: Domain Component
 - etc!

dn: cn=John Doe,dc=example,dc=com

cn: John Doe

givenName: John

sn: Doe

telephoneNumber: +1 123 456 789

mail: john@example.com

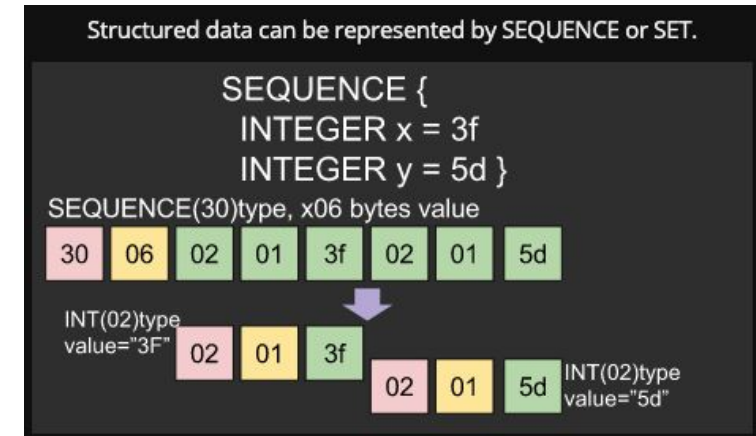
Certificate Structure: X.509

- Standard for information fields:
 - Version (v3) / serial number
 - Validity period
 - Subject / Alt. Names
 - Issuer
 - Key usages
 - Signature Algorithm
 - Key Fingerprints + Digital Signatures
 - Extensions...

Version Number	
Serial Number	
Signature Algorithm ID	
Issuer Name	
Validity Period	Not Before
	Not After
Subject Name	
Subject Public Key Info Public Key Algorithm Subject Public Key	
Issuer Unique Identifier (optional)	
Subject Unique Identifier (optional)	
Extensions (optional)	
Certificate Signature Algorithm	
Certificate Signature	

Certificate: encodings vs standards

- X.509 standard, but many possible encodings!
 - ASN.1 – abstract data type notation (X.690)
 - Basic/Canonical/Distinguished Encoding Rules
 - *DER* – “there is one and only one way to encode a message”
- File formats (most ASN.1-based):
 - PEM (GPG, SSH), p10/p8/p7 (PKCS #) etc.
- Other certificate standards:
 - SPKI
 - RFC 2440 (OpenPGP Message Format)
 - Card Verifiable Certificates - embedded devices



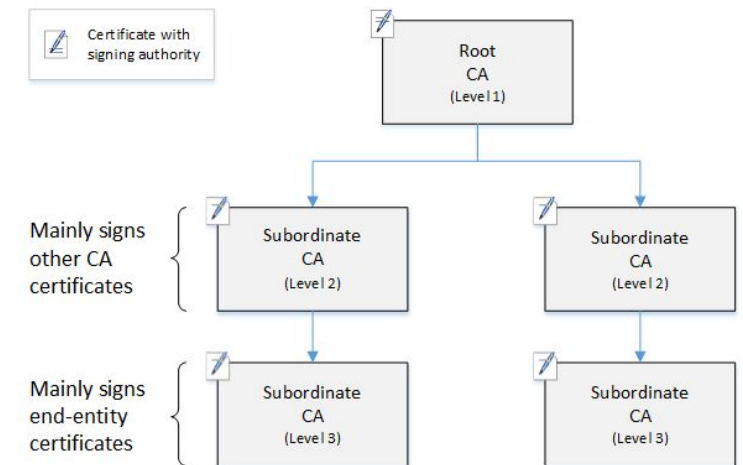
Integer value	BER encoding
0	02 01 00
127	02 02 00 7F
128	02 02 00 80
256	02 02 01 00
-128	02 01 80
-129	02 02 FF 7F

Obtaining Certificates


- Applicant generates PK + certificate signing request (CSR)
 - Public key + identification (CN, OU etc.), purpose & other fields
 - PKCS #10 standard 🙄
- Submits CSR to Certification Authority
 - E.g., governmental office / mail / web / automated protocols
 - **Must** be done via **trusted** channel!
- CA verifies the identity, signs & emits certificate
 - Gives digital file back to the applicant (via **untrusted** channels)
- Start using the certificate!
 - e.g., configure TLS server

Certificate Authorities

- Commercial / Non-Profit organisations implementing rigorous validation standards
 - Must secure their data (signing private keys)
 - Must maintain public trust
 - Must NOT sign off fraudulent identities!
- Hierarchical approach:
 - Leaf CA => Intermediate CAs (split roles) => trusted by Root CAs
- Trust the gatekeepers?
 - Mutual assurance: popular OS & browser vendors! + Certificate Transparency

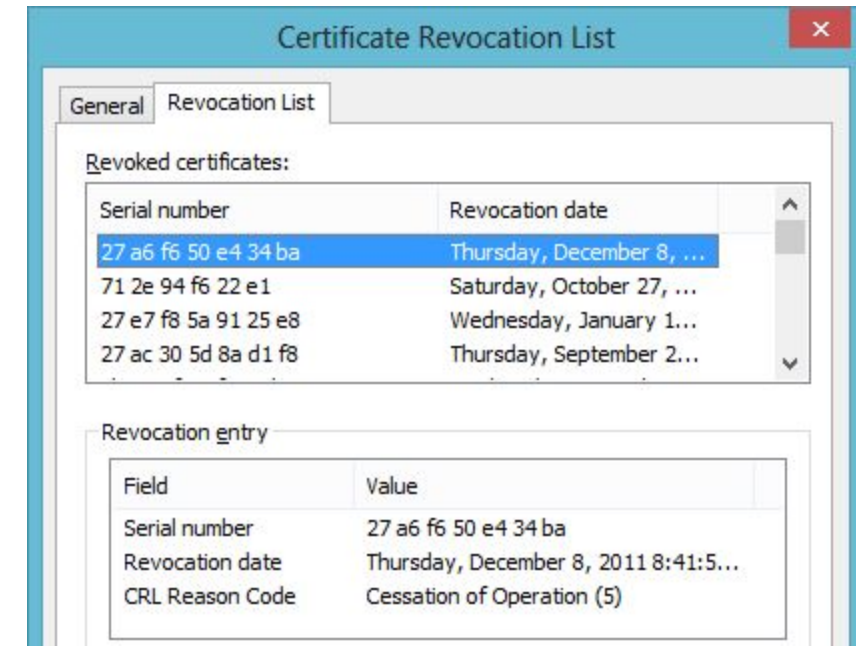


Self Certificate Authorities

- **Create your own CA (and delete all others):** government or 
- Step 1: Create a private key for the CA
- Step 2: Create Certificate of the CA
- Step 3: Add the CA certificate to the trusted root certificates
 - `sudo cp CA.crt /usr/local/share/ca-certificates`
- Step 4: Create a certificate for the webserver
- Step 5: Sign the certificate
- Step 6: Deploy the certificate
- Step 6: Done!

Certificate Revocation

- Server is compromised, private key stolen
 - Certificate valid until expiration date?
- CAs have another role: revocation
 - CAs can also be revoked 😈
- Certificate Revocation Lists (CRLs)
- Online Certificate Status Protocol (OCSP)



Certificate Revocation Lists (CRLs)

- CRL = An URL where revoked certificates are stored
 - Can be accessed via HTTPS, LDAP, FTP
- CRLs must be also signed by CA
- Someone (browsers) need to verify the list
 - What happens if the list is not available? (DoS on the browser)
- Reasons to revoke, hold, or unlist a certificate (RFC 5280)
 - unspecified (0)
 - keyCompromise (1)
 - cACompromise (2)
 - privilegeWithdrawn (9)

Online Certificate Status Protocol (OCSP)

- Protocol used to replace the (simple, but heavy) CRLs
- Requests are made per certificate, not the full list

```
CertID ::= SEQUENCE {  
    hashAlgorithm      AlgorithmIdentifier,  
    issuerNameHash      OCTET STRING,  
    issuerKeyHash       OCTET STRING,  
    serialNumber        CertificateSerialNumber  
}
```

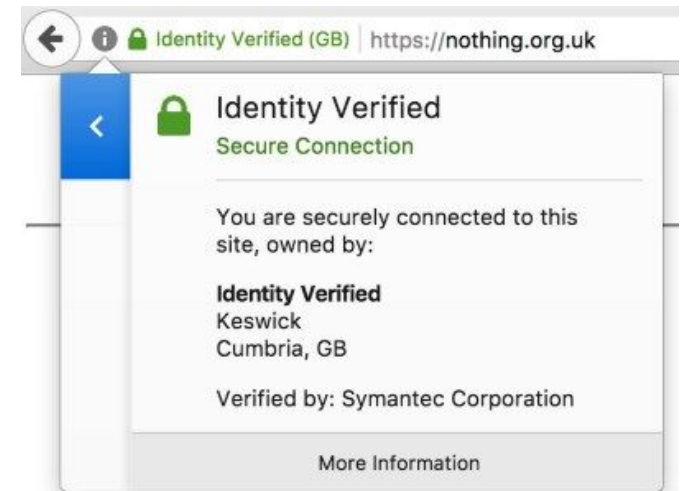
Authority Info (AIA)

Location	http://o.pki.goog/wr2
Method	Online Certificate Status Protocol (OCSP)
Location	http://i.pki.goog/wr2.crt
Method	CA Issuers

- Client - Server protocol
 - Server is specified in the certificate to be checked
- OCSP protocol uses ASN.1 format over HTTPS
- OCSP stapling:
 - Caching mechanism for the server to send the certificate status directly with the certificate

Certificate Usage & Validation

- Server vs Client/User (e.g., VPN authentication)
- Key Usage / constraints:
 - digital signatures, non-repudiation, certificate signature (for CAs), CRL signature, encryption etc.
- Validation level:
 - Domain Validation
 - Organisation Level
 - Extended Validation



Certificate Authority Types

- Government CAs (e.g, EU Digital Identity)
- Commercial CAs
 - GlobalSign, IdenTrust, Comodo, DigiCert, Verisign etc.
 - OS/Browser Root CAs lists: > 100 trusted authorities!
- Open-Source CAs:
 - Let's Encrypt: >50% market share!
 - Only Domain Validation :(fully automated!
- New players? Cross-Signing!
 - Let's Encrypt was signed by IdenTrust for backwards compatibility!

Lets Encrypt: ACME Protocol

- Automatic Certificate Management Environment by ISRG
- Automate CSR generation & validation:
 - CA challenges client with random nonce
 - Cert. client installs nonce on either server (HTTP) or DNS
 - CA queries server/DNS to check for that nonce
- Tools / libraries:
 - certbot, <https://letsencrypt.org/docs/client-options/>

Compromised CAs

- Supply Chain Attack: attack CAs / steal priv keys & forge certificates
- or: Untrustworthy CAs in browser databases (e.g., state controlled)
- Incidents:
 - Thawte (2008) – validation: register *sslcertificates@live.com* and obtain a rogue SSL certificate from Thawte for Microsoft's live.com!
 - DigiNotar (2011) – hacked, MitM for Iranian users, bankrupt
 - TurkTrust (2011) accidentally issues two intermediate CA certificates to subscribers
 - MCS Holdings (2015, China) issued certificates for Google domains 😊

Certificate Transparency

- *Who watches the watchers?*
- CA compromised... how to detect foul play?
 - Publish all issued certificates on a append-only ~~blockchain~~ public log!
- How?
 - SCR -> PreCertificate -> Send to logs -> Signed Certificate Timestamp
 - SCT – promise the certificate will be appended within time window -> send back to CA -> finally obtain valid certificate (with SCT embedded within)!
- Browsers require proof of SCT: Chrome / Safari

Alt. public key validation approaches

- Problem: domain validation not applicable
 - Email (on same domain, e.g. gmail.com)
 - Instant Messaging (user chat confidentiality)
- Self-signed certificates
- Out of band / manual **fingerprint** validation
- Trust on First Use (TOFU)
- Decentralized / Web of Trust!

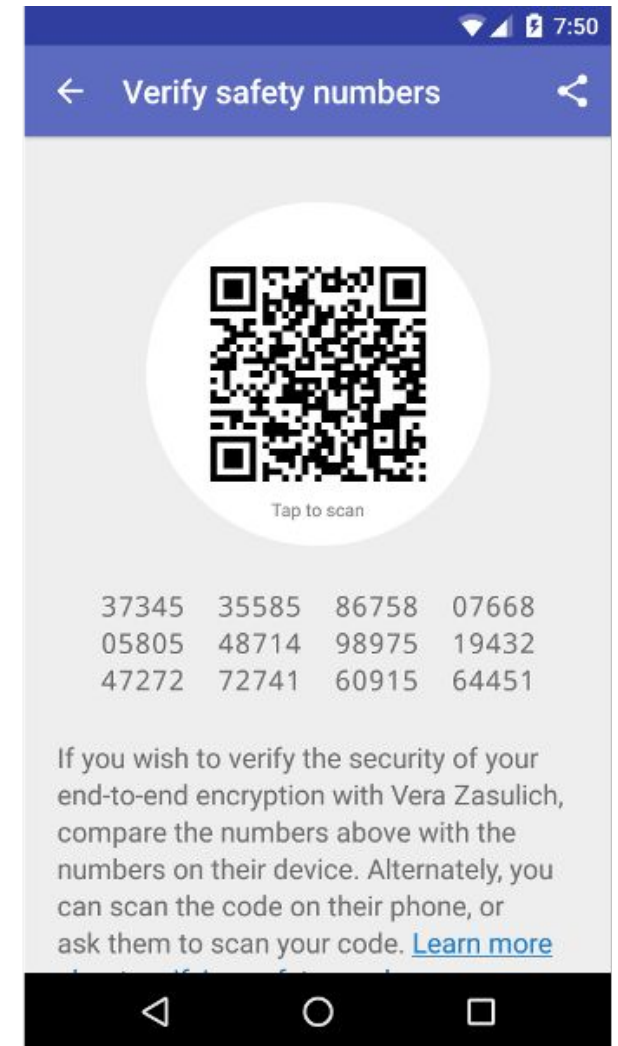
TOFU

- Trust/Check on First Use / *Leap of Faith*
- SSH: do you accept this key?
 - Key remains cached (~/.ssh/known_hosts)
- Compare fingerprints

```
[root@scala1 ~]# ssh root@10.11.0.90
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:2ZSDQI6JNjAleZph9MiK1gpuP6R7V6Uv3nngJO8oWVU.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending RSA key in /root/.ssh/known_hosts:27
RSA host key for 10.11.0.90 has changed and you have requested strict checking.
Host key verification failed.
[root@scala1 ~]#
```

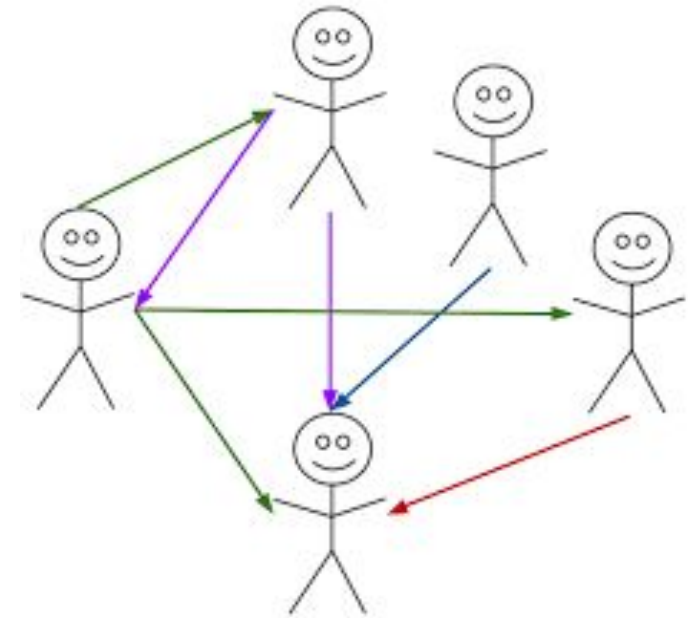
Off-The-Record Messaging

- Chat applications -> untrusted server
 - E.g.: Zucc / NSA / Russian gov.
- TOFU + fingerprint checking
 - QR codes FTW!
 - *who does this?*
- Example Applications:
 - XMPP (Jabber)
 - Signal / WhatsApp etc.



GPG/PGP – Web of Trust model

- Decentralized, ad-hoc management of keys
- Does not use X.509 => lightweight certificates
- Friends endorse a person
 - Key signing parties / conferences etc.
- Initial anchors? Trusted key servers ;)
- Trust scoring
 - Must be signed by fully-trusted or at least 3 marginally trusted keys
 - Trust doesn't propagate when path length > 5

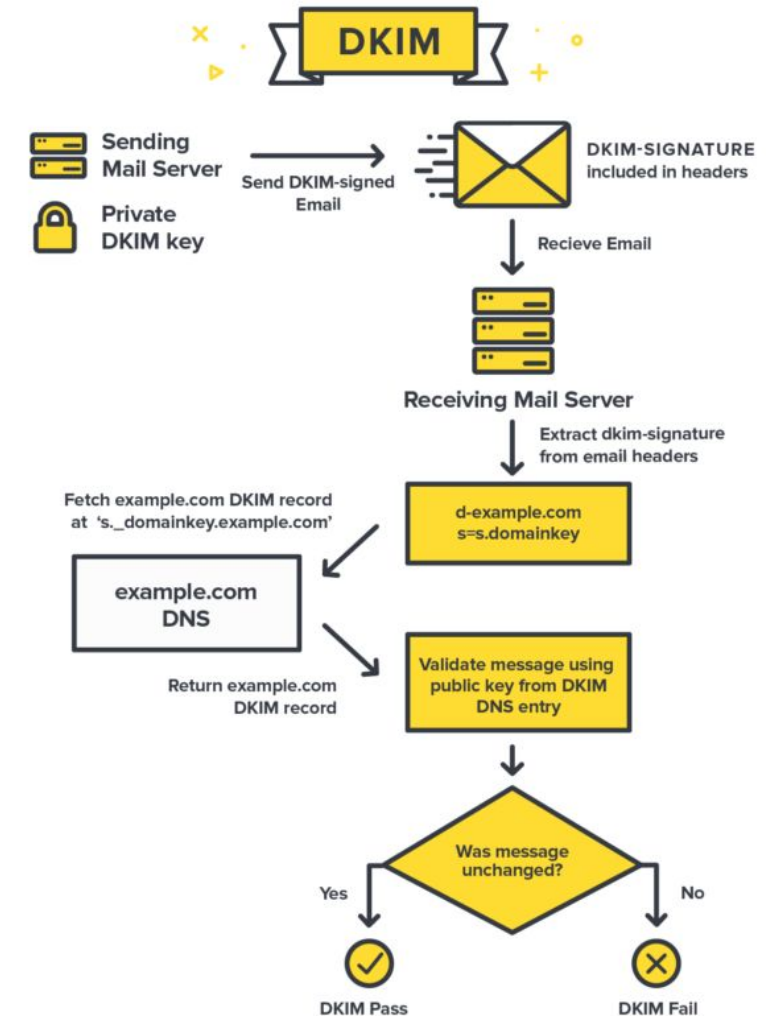


Secure Email

- Client - Mail Server: STARTTLS / IMAPS / SMTPS
- P2P / End to end encryption?
 - PGP/GPG, Secure/Multipurpose Internet Mail Extensions (S/MIME)
 - Public key usually sent as attachment (TOFU?)
- Problems: key provisioning, webmail...
- Outlook, Gmail, and Apple Mail support S/MIME
- Thunderbird has extensions for S/MIME & GPG
- Browser-based mail: extensions (WebPG)
- Provider-based encryption: ProtonMail

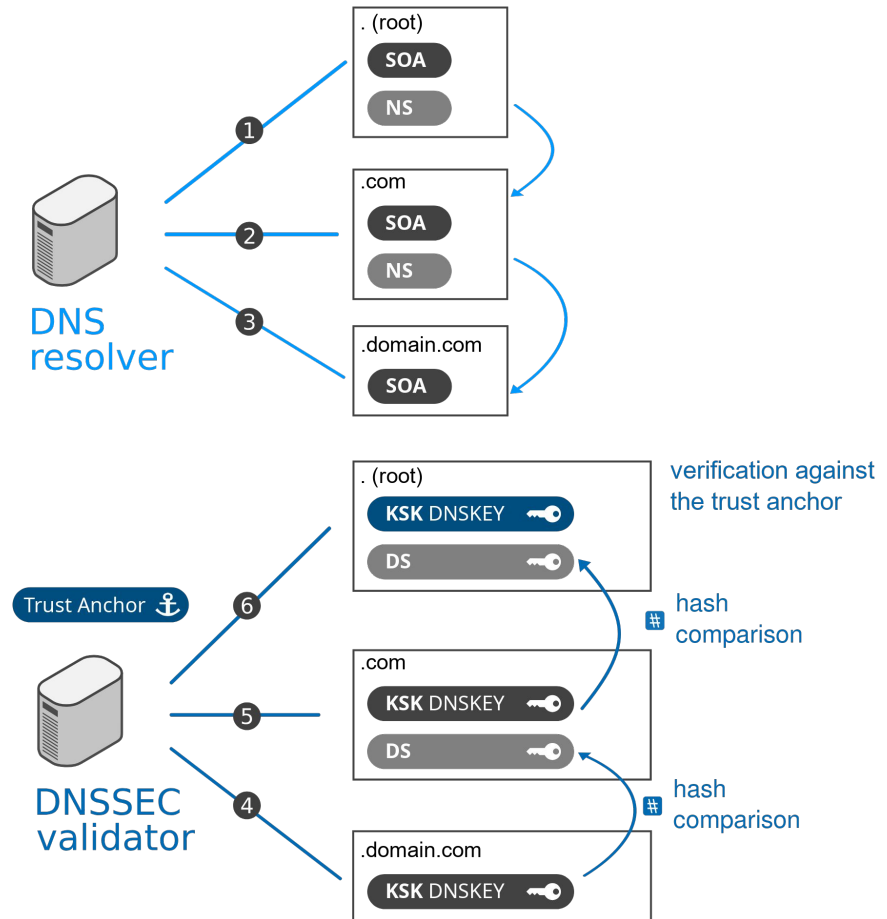
Email Server Authentication

- Prevent mail spoofing & spam
- DKIM & SPF: both use DNS records
- DomainKeys Identified Mail:
 - Public-key signature of emails originating from server
- Sender Policy Framework
 - IP addresses allowed to send using SMTP server
 - E.g.: allow marketing services (e.g., sendgrid / mailgun) to send mails using company's domains



DNSSEC

- DNS cache easily poisoned!
- Domain Name System Security Extensions:
 - Domain has priv/pub key (DNSKEY record) and signs all records
 - Parent zone authenticates your pubkey (and so on, recursively)
 - Root DNS zones' keys are trusted by resolvers (similar to Root CAs in browsers)
- Actually, the recursive resolve process is a bit more complicated than this (:



References

- [1] <https://people.scs.carleton.ca/~paulv/toolsjewels/TJrev1/ch8-rev1.pdf>
- [2] <https://letsencrypt.org/how-it-works/>
- [3] <https://www.sectigo.com/resource-library/understanding-the-different-types-of-certificate-authorities>
- [4] <https://scotthelme.co.uk/cross-signing-alternate-trust-paths-how-they-work/>
- [5] <https://signal.org/blog/safety-number-updates/>
- [6] <https://certificate.transparency.dev/howctworks/>