Chapter 8

Public-Key Certificate Management and Use Cases

The official version of this book is available at
https://www.springer.com/gp/book/9783030336486

ISBN: 978-3-030-33648-6 (hardcopy), 978-3-030-33649-3 (eBook)

version: 25 Sept 2019

Chapter 8

Public-Key Certificate Management and Use Cases: TLS, Secure Email

This chapter explains certificate management and public-key infrastructure (PKI), what they provide, technical mechanisms and architectures, and challenges. Two major applications are also considered here as examples: TLS as used in HTTPS for secure browser-server communications, and end-to-end encrypted email. Additional applications include SSH and IPsec (Chapter 10), DNSSEC (Chapter 11), and trusted computing.

In distributed systems, cryptographic algorithms and protocols provide the foundations for access control to remote computing resources and data services, and for authorization to change or store data, and to remotely execute commands. Authentication is a common first step in authorization and access control. When passwords are used for remote authentication, they travel over a channel itself secured by authentication and confidentiality based on cryptographic keys. These keys protect not only data in transit but also data at rest (stored). *Key management*—the collection of mechanisms and protocols for safely and conveniently distributing such keys—includes managing not only session keys per Chapter 4, but public keys (as discussed herein) and their corresponding long-term private keys.

8.1 Certificates, certification authorities and PKI

We first discuss certificates and certification authorities, and then give a quick overview of public-key infrastructure.

INTEGRITY OF PUBLIC KEYS. Public keys are used in cryptographic algorithms and protocols.[1] As their name suggests, they need not be kept secret—a defining property is that knowing a public key does not allow deduction of the corresponding private key. However, the *authenticity* (and related *integrity*) of a public key is essential for security—by this, we mean knowing to whom a public key "belongs", since that is the party assumed

[1] Chapter 2 provides background on public-key cryptography, and briefly introduces certificates.

to know, and protect, the corresponding private key. The danger is that if the encryption public key of an intended recipient *B* is substituted by that of an opponent, the opponent could use their own private key to recover the plaintext message intended for *B*.

PUBLIC-KEY CERTIFICATES. A *public-key certificate* is used to associate a public key with an owner (i.e., the entity having the matching private key, and ideally the only such entity). The certificate is a data structure that *binds* a public key to a named `Subject`, by means of a digital signature generated by a *trusted third party* called a *Certification Authority* (CA). The signature represents the CA's assertion that the public key belongs to the named `Subject`; having confidence in this assertion requires trust that the CA making it is competent and reliable on such statements. Any party that relies on the certificate—i.e., any *relying party*—places their trust in the issuing CA, and requires the corresponding valid public key of that CA in order to verify this signature. Verifying the correctness of this signature is one of several steps (Section 8.2) that the relying party's system must carry out as part of checking the overall validity of the target public-key certificate.

NAMES IN CERTIFICATES. The certificate fields `Subject` (owner) and `Issuer` (signing CA) in Table 8.1 are of data type `Name`. `Name` is a set of attributes, each a pair <attribute name, value>. Collectively, the set provides a unique identifier for the named entity, i.e., serves as a *distinguished name* (DN). Commonly used attributes include: `Country` (C), `Organization` (O), `Organizational Unit` (OU), and `Common-Name` (CN). Examples are given later in the chapter (Figures 8.9 and 8.11).

Field name	Contents or description
`Version`	X.509v3 or other versions
`Serial-Number`	uniquely identifies certificate, e.g., for revocation
`Issuer`	issuing CA's name
`Validity-Period`	specifies dates (`Not-Before`, `Not-After`)
`Subject`	owner's name
`Public-Key` info	specifies (`Public-Key-Algorithm`, `Key-Value`)
extension fields (optional)	`Subject-Alternate-Name/SAN-list`, `Basic-Constraints`, `Key-Usage`, `CRL-Distribution-Points` (and others)
`Signature-Algorithm`	`(algorithmID, parameters)`
`Digital-Signature`	signature of `Issuer`

Table 8.1: X.509v3 public-key certificate fields.

CERTIFICATE FIELDS. Beyond the public key, `Subject` and `Issuer` names, and CA signature, a certificate contains other *attribute* fields that allow proper identification and safe use of the public key (Table 8.1). These include: format version, serial number, validity period, and signature algorithm details. The public-key field has two components, to identify the public-key algorithm and the public-key value itself. X.509v3 certificates, which are the certificates most commonly used in practice, have both these basic fields and *extension fields* (Section 8.2). The CA signature is over all fields for integrity protection, i.e., the hash value digitally signed encompasses all bits of all fields in the certificate.

CA checks before issuing certificate. Before a CA issues a certificate to a requesting party, it is expected to exercise due diligence. A CA is vouching not for the character or integrity of the entity named in the certificate, but rather for an association between an entity name and a public key. Related to a certificate's `Public-Key` and `Subject` or `Subject-Alternate-Name`, three aspects deserve special attention:

1. Evidence of knowledge of the corresponding private key. A malicious party should not be able to acquire a certificate associating its name with another entity's public key. The requesting party should thus be required to provide a *proof of knowledge* of the private key, e.g., by the CA sending a fresh challenge and verifying a response, wherein the CA uses the public key and the requesting party uses the private key.
2. Evidence of ownership or control of computer-addressable identities related to the `Subject` field in the pending certificate. For example, control of an asserted domain name, email address or phone number should be demonstrated to the CA.
3. Confirmation of asserted natural-world names (for high-quality certificates). If an organization name is asserted, the CA should carry out cross-checks to confirm the requesting party is legitimately affiliated with, or authorized by the organization to acquire the pending certificate; if an individual name is asserted and intended to represent a natural-world person (rather than an explicitly identified *pseudonym*), then suitable personal identification may be requested.

The extent of these and other checks made depends on the operational policy of the CA. For TLS certificates, three grades of certificate result (Section 8.5).

Acquiring a certificate. Protocols have been standardized for an end-entity to request a certificate from a CA (exercise below). An end-entity sends the CA a *certification request* including a DN, public key, and optional additional attributes. This is typically preferred over the CA itself generating the end-entity's public-private key pair, in which case the CA must be trusted not to disclose or abuse the private key.

‡**Exercise** (Certificate management protocol). Summarize the protocols of RFC 4210, by which an end-entity requests and receives a certificate from a CA. (Hint: [1]. This RFC allows certificate requests per the earlier PKCS #10 of RFC 2986 [37].)

PKI. A *public-key infrastructure* (PKI) is a collection of technologies and processes for managing public keys, their corresponding private keys, and their use by applications. Its primary end-goal is to facilitate use of long-term keys used to authenticate entities, and to enable establishment of authenticated session keys. In this way, PKI facilitates encryption, data integrity, and also digital signatures for entity authentication, data origin authentication and (in theory, but far less commonly in practice) non-repudiable, legally recognized signatures. PKI involves (Fig. 8.1):

1. data structures related to key management (e.g., certificates, formatted keys);
2. use of cryptographic toolkits and related methods for automating key management;
3. architectural components such as CAs and (public-key) *certificate directories*; and
4. procedures and protocols for approving, acquiring and updating keys and certificates.

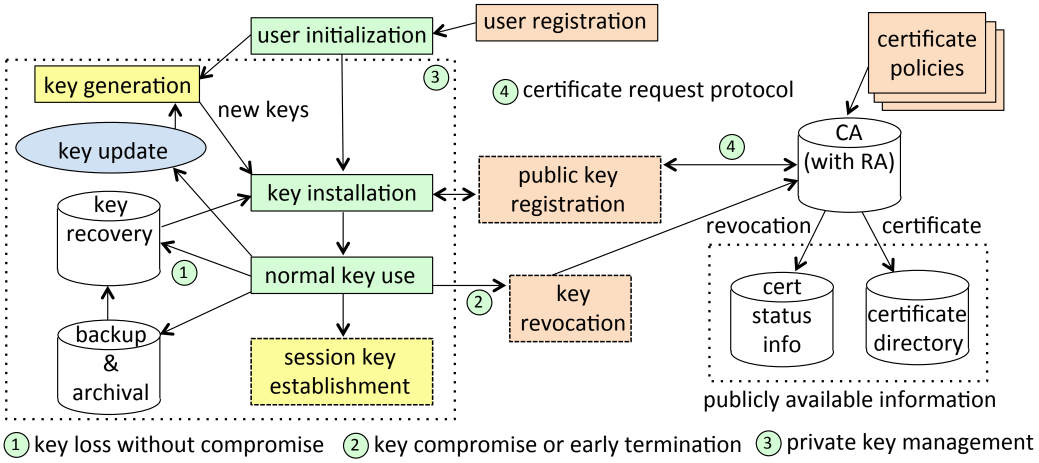


Figure 8.1: PKI components and lifecycle.

Standardization of PKI elements, e.g., X.509v3 *certificate profiles* (describing common certificate fields and how they are used), has aided interoperability and use of supporting components and software libraries across vendors and products. PKI support for public-key cryptography improves automation, scalability, security and convenience compared to infrastructures based on symmetric keys only, and older practices (manual sharing, manual entry, and hard-coding of symmetric keys and passwords).

‡ADDITIONAL PKI ASPECTS. A challenging aspect often overlooked is management of long-term private keys. If stored encrypted under keys derived from user-chosen passwords, they are at risk to offline password-guessing attacks (Chapter 3). Lifecycle support for non-repudiable signatures adds significant complexity (beyond common capabilities, and e.g., requiring notary services), to reconstruct time-relevant revocation information. CAs may also use a Registration Authority (RA) to facilitate certificate requests.

‡**Exercise** (PKI components and lifecycle management). Figure 8.1 outlines major PKI components. Summarize the tasks each addresses (hint: [35, Fig. 13.10 on p. 579]).

8.2 Certificate chain validation and certificate extensions

VALIDATING CERTIFICATES AND CERTIFICATE CHAINS. CAs issue certificates for end-entities, e.g., human users in the case of email certificates, web servers in the case of TLS certificates. A CA may also issue a certificate for the public key of another CA, sometimes called an *intermediate CA*, e.g., when the first CA is a trust anchor or atop a hierarchy (Section 8.4). This results in the concept of a *certificate chain* (Fig. 8.2). Before the public key in a certificate is relied on for some intended purpose, the relying party should *validate* the certificate, i.e., check to ensure that “everything is in order”. The steps for validating a certificate include checking that the target certificate:

1. has not expired, and the current date is in the range [`Not-Before`, `Not-After`];

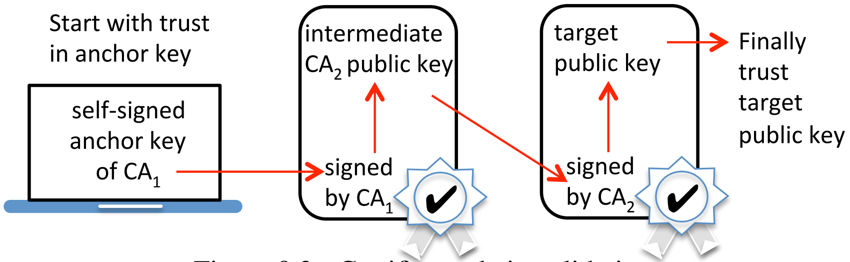

Start with trust in anchor key
self-signed anchor key of CA_1
intermediate CA_2 public key
signed by CA_1
target public key
signed by CA_2
Finally trust target public key

Figure 8.2: Certificate chain validation.

2. has not been revoked (Section 8.3);
3. has a signature that verifies (mathematically), using the signing CA's public key;
4. is signed by a CA whose public key is (available and) itself trusted;
5. has a `Subject` or `Subject-Alternate-Name` matching the semantics of use. For example, if it is supposedly a TLS certificate from a browsed-to domain, the domain name in the certificate should match the URL domain the browser is visiting. If the certificate is for encrypting email to party *B*, the email address that the mail client believes corresponds to *B* should match that given in the certificate.[2]
6. use is consistent with all constraints specified in certificate extension fields or policies (e.g., path length, key usage restrictions, name constraints—explained on page 220).
7. if not directly signed by a *trust anchor* CA, then a valid chain of certificates from a trust anchor to the target certificate must be available, with all the above steps checked for every certificate in the chain (Fig. 8.2). Trust anchors are defined on page 219.

Example *(Passport analogy to certificate policy constraints).* Countries issue passports. Most other countries recognize these (and in new e-passports, now verify the issuing government using public keys). Recognizing that a passport validly identifies a citizen of another country does not obligate a country to allow entry of that individual. Independent of such recognition, one country can decide to disallow entry of travellers from another.

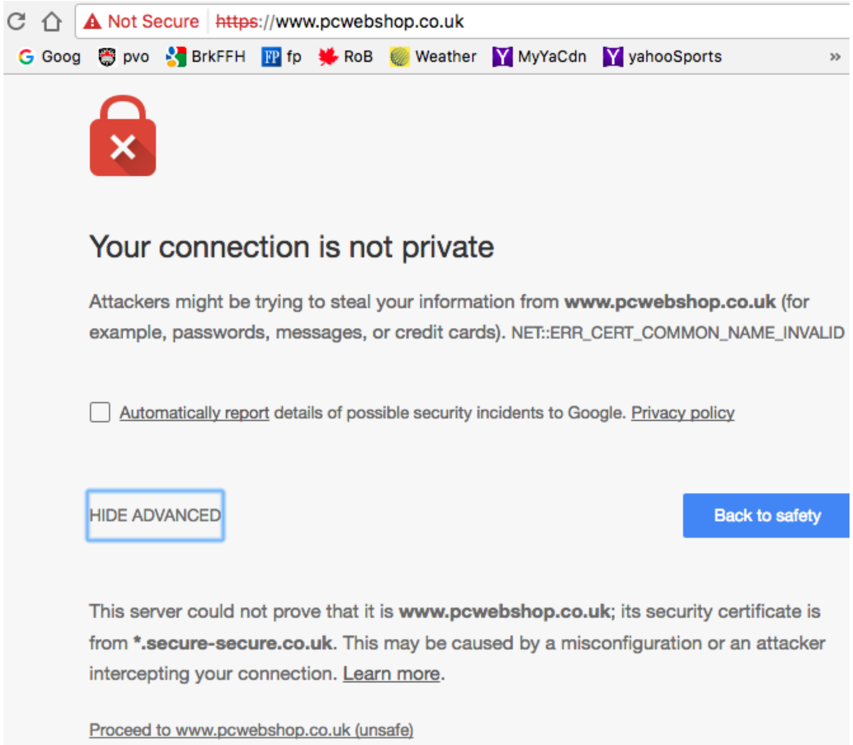
OUT-OF-BAND CHANNELS & CHECKING FINGERPRINTS. Trust is often initially established by information sent over a channel that by assumption, an attacker does not have access to (if confidential) or cannot alter (if integrity is required)—sometimes called an *out-of-band* channel. The term also arises in a process called checking *fingerprints*. An example is retrieving a public key over an untrusted Internet channel such as email or HTTP, locally computing a cryptographic hash of the retrieved data, and cross-checking the data's integrity by comparing the computed hash to a hash value (believed to be the authentic one) obtained over an out-of-band channel.[3] Such hash values may be represented as hexadecimal strings, or images uniquely derived from them. The out-of-band channel might be paper via postal mail or courier, or data exchanged in person, by voice over phone or from an HTTPS web site. The term *out-of-band* derives from telephony, where *in-band signaling* means sending control data over the same channel as voice data.

[2] This is an example of principle P19 (REQUEST-RESPONSE-INTEGRITY).

[3] This combines P17 (TRUST-ANCHOR-JUSTIFICATION) and P18 (INDEPENDENT-CONFIRMATION).

SELF-SIGNED CERTIFICATES & BROWSER TRUST ANCHORS. A public-key certificate is commonly validated using an in-hand trusted public key to verify the certificate's signature. In contrast, *self-signed* certificates are signed by the private key corresponding to the certificate's own public key. This does not allow deriving trust in one public key from another, but serves as a convenient structure for packaging a public key and related attributes. Trust in a self-signed certificate should be established by a reliable out-of-band channel. Some browsers have a *trusted certificate store* of self-signed certificates of a large number of established CAs, vetted by the browser vendor; other browsers rely on a similar store maintained by the host operating system, and either option may use a small set stored locally, with a larger set hosted online by the vendor to dynamically augment the local set. This dictates which TLS (server) certificates the browser, on behalf of the user, will recognize as valid. Such CA public keys relied on as pre-trusted starting points for certificate chains are called *trust anchors*.

ACCEPTING UNTRUSTED CERTIFICATES. A browser visiting a web site may be sent a self-signed certificate (Figure 8.3). The browser software may reject it, or present a dialogue allowing the user to accept it as "trusted" despite the software not recognizing



⚠ Not Secure ~~https~~://www.pcwebshop.co.uk

Your connection is not private

Attackers might be trying to steal your information from **www.pcwebshop.co.uk** (for example, passwords, messages, or credit cards). NET::ERR_CERT_COMMON_NAME_INVALID

□ Automatically report details of possible security incidents to Google. Privacy policy

This server could not prove that it is **www.pcwebshop.co.uk**; its security certificate is from ***.secure-secure.co.uk**. This may be caused by a misconfiguration or an attacker intercepting your connection. Learn more.

Proceed to www.pcwebshop.co.uk (unsafe)

Figure 8.3: Self-signed site certificate—browser warning (Chrome 56.0.2924.87). The "Back to safety" tab discourages clicking-through to the site despite the browser being unable to verify the certificate chain (this happens when one or more certificates are signed by CAs unrecognized by the browser, i.e., not verifiable using the browser's trust anchors).

it as such—in this case ideally also presenting the user information allowing a certificate fingerprint check as discussed (Fig. 8.10, page 232, shows two fingerprints). Users may accept the certificate without bothering to check, or may have insufficient information or understanding to check properly—but if they accept, they do so at their own risk, even if they do not understand this or the consequences of doing so.

An analogous situation arises if the received certificate is CA-signed but the receiving software has no trust anchor or chain allowing its programmatic verification. Again, the client application may be programmed to allow users to accept ("trust") such certificates "by manual decision". This violates a basic usable security[4] principle—users should not be asked to take decisions that they do not have sufficient information to make properly—but it is a common shortcut when software designers don't have better design ideas.

Trust on first use (TOFU). If a self-signed certificate is accepted (relied on as "trusted") the first time it is received from a remote party, without any cross-check or assurance that it is authentic, this is called *trust on first use* (TOFU). To emphasize the risk, and lack of cross-check, it is also called *blind TOFU* or *leap-of-faith* trust. Some software interfaces ask the user whether the key should be accepted for one-time use only, or trusted for all future uses; the latter is sometimes assumed silently with the public key stored (within its certificate packaging), associated with that party, application or domain, and checked for a match (*key continuity*) on subsequent uses. If an active attacker provided a forged certificate on this first occurrence, the gamble is lost; but otherwise, the gamble is won and subsequent trust is justified. If a fingerprint is cross-checked once before first use, rather than "TOFU" we may call it *check on first use* (COFU).

Trust anchor justification. TOFU is common when SSH (Chapter 10) is configured for authentication with user password and server public key. On first visit to an SSH server, the SSH client receives the server public key and is given an option to accept it. If the user accepts (after optionally cross-checking its fingerprint), the client stores the public key (for future use) and uses it to establish a secure channel; a user-entered password is then sent over this channel for user authentication to the server. On return visits to this server, the newly received public key is (silently) cross-checked with the stored key. This highlights a critical point: many PKI tools are designed to fully automate trust management after initial keys are set as trusted, proceeding thereafter without user involvement, silently using any keys or trust anchors accepted or configured in error. The importance of attention and correctness in such manual trust decisions motivates principle P17 (TRUST-ANCHOR-JUSTIFICATION). This applies when accepting keys or certificates as trusted, especially by non-technical users, in applications such as browsers (HTTPS certificates), secure email clients (PGP, S/MIME certificates), and SSH as above.

X.509v3 extensions. Version 3 of the X.509 certificate standard added *certificate extension* fields. These are marked either *critical* or *non-critical*. An older system may encounter an extension field that it is unable to interpret. If the field is marked non-critical, the system can ignore the field and process the rest of the certificate. If a field is marked critical and a system cannot process it, the certificate must be rejected. Some examples of

[4] Usable security is discussed in Section 9.8.

extensions follow.

- `Basic-Constraints`: this extension has the fields `(cA, pathLenConstraint)`. The first field is boolean—TRUE specifies that the public key is for a CA, FALSE specifies that the key is not valid for verifying certificates. The second field limits the remaining allowed certificate chain length. A length of 0 implies the CA can issue only end-entity, i.e., *leaf* certificates (this CA key cannot be used to verify chain links).
- `Key-Usage`: this specifies allowed uses of a key, e.g., for signatures, encryption, key agreement, CRL signatures (page 222). A separate extension, `Extended-Key-Usage`, can specify further key uses such as *code signing* (vendor signing of code to allow subsequent verification of data origin and integrity) and TLS server authentication.
- `Subject-Alternate-Name`: this may include for example an email address, domain name, IP address, URI, or other name forms. If the `Subject` field is empty (which is allowed), the alternate name must be present and marked as a critical extension.
- `Name-Constraints`: in CA-certificates (below), these allow CA control of `Subject` names in subsequent certificates when using hierarchical *name spaces*. By specifying prefixes in name subtrees, specified name spaces can be excluded, or permitted.

CROSS-CERTIFICATE PAIRS. ITU-T X.509 standardized a data structure for a pair of *cross-certificates* between CAs, each issuing a certificate for the other's public key—one `issued-to-this-CA`, one `issued-by-this-CA`. For example, a cross-certificate pair can allow CAs at the roots of two hierarchies (Section 8.4) to enable secure email between their communities. This data structure can aid discovery and assembly of certificate chains. Single (unilateral) cross-certificates are also possible strictly within one hierarchy, but in this case, *CA-certificate* is a less confusing term for one CA issuing a certificate to another. Constraints placed on cross-certificates and CA-certificates via certificate extensions take on greater importance when extending trust to an outside community.

Exercise (Transitivity of trust). Certificate chains, depending on constraints, treat trust as if it is transitive. Is trust transitive in real life? Consider the case of movie recommendations from a friend. (On what subject matter do you trust your friend? From whom do you seek legal relief if something goes wrong in a long trust chain?)

8.3 ‡Certificate revocation

Certificates have a predefined expiration date, from their validity field. A typical period is 1–2 years. Analogous to conventional credit cards, the validity period may be terminated ahead of time, i.e., the certificate can be *revoked*. In centralized systems where certificates are signed by CAs, it is expected that the CA issuing a certificate is responsible for making information about revoked certificates available to relying parties; the issuing party is recognized as the revocation authority for that certificate.

REVOCATION REASONS. For public-key certificates, the most serious *revocation reason* is the compromise, or suspected compromise, of the `Subject`'s private key. Other reasons may include that the key has been superseded by another key prior to the planned expiry; the key owner is discontinuing use of the key; or the `Subject` (owner) changed job titles or affiliation and requires a new key for the new role.

We next discuss some of the main approaches used for revoking certificates.

Method I: Certificate revocation lists (CRLs). A CA periodically issues (e.g., weekly, perhaps more frequently) or makes available to relying parties in its community, a signed, dated list of serial numbers of all unexpired-but-revoked certificates among those it has issued. The CRL may be sent to members of a defined community (*push model*), or published at an advertised location (*pull model*). Individual certificates may themselves indicate the retrieval location. An issue with CRLs is that depending on circumstances, their length may become cumbersome. Shortening certificate validity periods shortens CRLs, since expired certificates can be removed from CRLs.

Method II: CRL fragments—partitions and deltas. Rather than publishing full CRLs, several variations aim to improve efficiency by using CRL fragments, each a dated, signed sublist of serial numbers. *CRL distribution points*, also called *partitioned CRLs*, break full CRLs into smaller pieces, e.g., corresponding to predefined serial number ranges. Different ranges might be retrieved from different locations. Different distribution points might be used for different categories of revocation reasons. A CRL distribution point extension field (in the certificate) indicates where a relying party should seek CRL information, and the method (e.g., LDAP, HTTP).

In contrast, the idea of *delta CRLs* is to publish updates to earlier lists (from the same CA); relying parties accumulate the updates to build full CRLs. When the CA next issues a consolidated CRL, subsequent updates are relative to this new, specified base list. To offload the effort required to assemble and manage delta CRLs, this variation may be supported by CRL aggregator services.

Method III: Online status checking. In online-checking methods such as the *online certificate status protocol (OCSP)*, relying parties consult a trusted online server in real time to confirm the validity status of a certificate (pull model). The appeal is in obtaining a real-time response—ideally based on up-to-date information (a real-time response is not necessarily based on fresh status information from the relevant CA; it might be no fresher than a CRL). In a push-model variation called *OCSP-stapling*, certificate holders frequently obtain signed, timestamped assertions of the validity of their own certificates, and include these when providing certificates to relying parties (e.g., when TLS servers send certificates to browsers).

Compromise timeline: from compromise to visibility. CRLs require no OCSP-style online revocation service, but suffer delays between when a revocation is made, and when a relying party acquires that knowledge. Heavy focus on the urgency to instantaneously broadcast revocation information may reduce legal liability, but overlooks other aspects. Consider the event timeline of a private key compromise in Figure 8.4.

Method IV: Short-lived certificates. This approach seeks to avoid the need for revocation entirely, by instead issuing certificates with relatively short validity periods—e.g., consider 1–4 days. The idea is that the maximum exposure window of a short-lived certificate is its full validity period, which is perhaps similar to or less than the exposure window of alternate methods. A drawback is the overhead of frequently

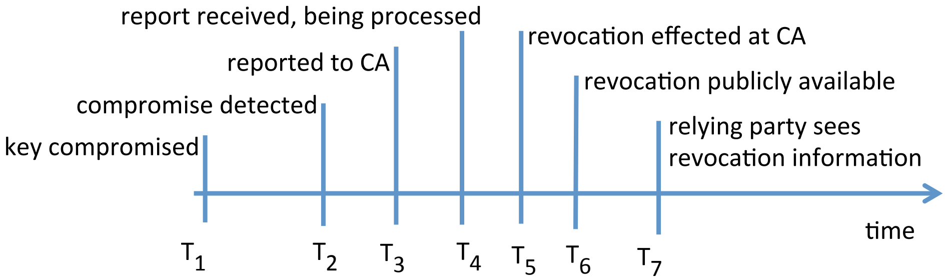

report received, being processed
reported to CA
compromise detected
key compromised
revocation effected at CA
revocation publicly available
relying party sees revocation information
time
T_1
T_2
T_3
T_4
T_5
T_6
T_7

Figure 8.4: Certificate revocation timeline: from compromise to visibility. Possible delays at T4 are mechanism-dependent, e.g., CRLs are typically issued at periodic intervals. A benefit of OCSP mechanisms (over CRLs) is to remove the T6-to-T7 delay.

re-issuing certificates. In the limit, short-lived certificates are created on-demand, at the expense of real-time contact with the authority who speaks for the key's validity, and the full-time load this places on that authority.

Method V: Serving trusted public keys directly. Continuing this line of thought leads to considering entirely eliminating not only revocation, but possibly even signed certificates, instead relying on a trusted key server to serve only valid keys.[5] This approach is best suited to a closed system with a single administrative domain; a real-time trusted connection to the server authoritative on the validity of each target public key requires relying parties have keying relationships with all such servers (or that one server acts as a clearinghouse for others)—raising key management issues that motivated use of certificates and related trust models in the first place. This approach also increases load on servers and availability requirements, compared to end-entities interacting with a trusted server only infrequently for new certificates. Thus significant tradeoffs are involved.

Revoked certificates: CA vs. end-entity. Both CA and leaf certificates can be revoked. Consider a CA issuing (a leaf) TLS certificate for a server, to secure browser-server connections. The server may have one or more TLS certificates from one or more CAs. One or all of these server certificates may be revoked. The certificate of the CA signing these certificates may also be revoked. These would all be distinct from the certificate of an end-user being revoked. (Recall that TLS supports mutual authentication, but in practice is used primarily for unilateral authentication of the server to the browser, i.e., the server presents its certificate to the browser.) X.509v3 standards include *Certification Authority revocation lists* (CARLs), i.e., CRLs specifically dedicated to revoked CA certificates; proper certificate chain validation includes revocation checks on CA keys throughout the chain, excluding trust anchors. This leaves the question of how to handle revocation of trust anchors, which albeit rare, may be by separate means, e.g., modification of trusted certificate stores in browsers or operating systems by software updates, and dynamic or manual changes to such stores.

Denial of service on revocation. A standard concern in certificate revocation is denial of service attacks. If a request for revocation information is blocked, the

[5] For related discussion of *public-key servers*, see Fig. 8.14 on page 237.

relying party's system either fails closed (deciding that the safest bet is to treat the certificate as revoked), or fails open (assumes the certificate is unrevoked). In the latter case, which violates principle P2 (SAFE-DEFAULTS), an attacker blocking revocation services may cause a revoked certificate to be relied on.

8.4 CA/PKI architectures and certificate trust models

This section explains how trust relationships between CAs, and the use of trust anchor lists, define PKI models suitable for TLS, secure email, and other applications.[6]

TRUST MODEL DEFINITION. There are many different meanings of trust and trust models. For our purposes, a *certificate trust model* is a system—its design, procedures, and rules as instantiated by software and other processes—by which applications recognize public-key certificates as valid, and if so, the allowed uses of their public keys as dictated by certificate fields or implied by relying systems. Of course such trust models, as technical mechanisms, cannot determine whether entities are trust*worthy*.

MODEL PHILOSOPHY VS. TECHNICAL ABILITY. It is helpful to think of a trust model as constraining a technology to implement a particular philosophy or achieve specific goals. Making a set of arbitrarily configurable software components available does not define a trust model; useful models facilitate specific purposes or applications. "What trust model is best?" is not a well-defined question. It is more useful to ask: "For a given application (with objectives X, Y and Z), what trust model is most suitable?" The answer differs by application, goals, and to whom the system is designed to give control.

‡TOOLBOX SHAPED TO MEET GOALS. Data structures associated with standard ITU-T:X.500, X.509 certificates, and customizations via IETF RFCs provide a toolbox allowing different communities (or applications) to build customized trust models suiting specific needs and ecosystems. Different trust models should be expected for automotive manufacturers (e.g., the use of VPN/IPsec technology by the Automotive Network eXchange/ANX), federal governments (e.g., the S/MIME-based U.S. Bridge CA secure email project), and business-to-consumer applications (e.g., TLS-based information exchanges between banks and online customers). PKI infrastructures intended for business-to-business partners may be underpinned by pairwise or network-wide formal business contracts and proprietary software; web-based e-commerce, on the other hand, may involve commodity browsers supporting TLS and click-wrap agreements that few read.

We discuss various trust models with examples to build our understanding—single-CA communities, linking them, CA hierarchies, linking multiple hierarchies, and finally hybrid models offering finer-grained configurability at the price of greater complexity.

MODEL I (BASE): SINGLE-CA DOMAINS AND LINKING THEM. The simplest systems involve single-CA domains (i.e., no hierarchies or intermediate CAs) with no trust connections between the distinct communities. Each is a single closed system with respect to trust, although possibly a very large administrative domain. This is the model of

[6] Instructors who are time-constrained may wish to focus on Model IV, as background for Section 8.5.

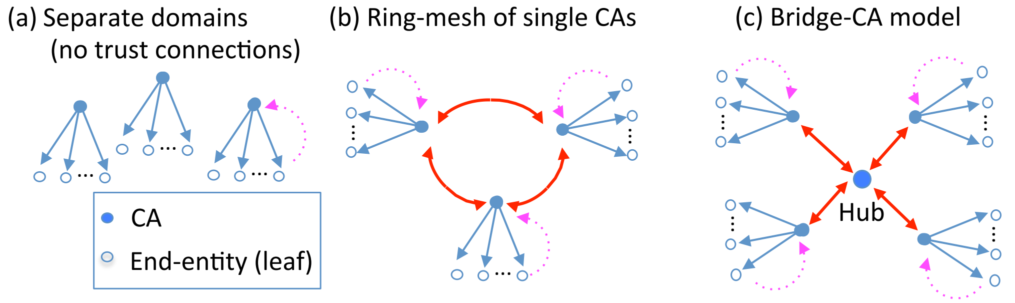


Figure 8.5: Model I: Single-CA systems (a) and linking them. Solid arrow points from certificate issuer to certificate subject. Double arrow indicates CA-certificates in both directions (i.e., cross-certificate pair). Dotted arrow indicates trust anchor. Case (b) shows that for $n = 3$ CAs, a ring of cross-certificate pairs (each pair of CAs signing a certificate for the public key of the other) is the same as a *complete network* (all CAs directly connected to all others). For $n \geq 4$, options include maintaining a ring structure (with each CA cross-certified only with immediate neighbors in a ring structure), or a complete network (all CAs pairwise cross-certified with all others). The hub-and-spoke model (c) reduces the inter-connect complexity from order n^2 to n.

current end-to-end secure *instant messaging* systems (e.g., WhatsApp Messenger). Figure 8.5 illustrates simple topologies for linking single-CA domains.

Example *(Linking single-CA systems).* Consider an enterprise company with three divisions in distinct countries. Each division administers a CA for in-country employees. Each end-entity is configured to have as trust anchor its own CA's key; see Figure 8.5(a). To enable entities of each division to trust certificates from other divisions, all being equal trusted peers, each pair of CAs can create certificates for each other, as indicated by double-arrows in Figure 8.5(b); the resulting *ring-mesh of single CAs* connects the formerly disjoint single-CA systems. As the caption notes, an $n = 3$ ring-mesh of CAs is a complete network, but at $n = 4$ CAs, the situation becomes more complex due to combinatorics: the number of CA pairs (4 choose 2) is now 6, and as a general pattern grows as the square of n. While direct cross-certifications between all CAs that are close business partners may still be pursued and desirable in some cases, it comes at the cost of complexity. This motivates an alternative: a bridge CA.

BRIDGE CA. The *bridge CA trust model*, also known as a *hub-and-spoke model*, is an alternative for large sets of equal peers or trading partners (as demonstrated in the U.S. Federal Bridge CA project, above). A dedicated bridge CA or hub node is introduced specifically to reduce the cross-connect complexity. Figure 8.5(c) shows this with single-CA subsystems; Model III's multi-CA subsystems can likewise be bridged. Note that the bridge CA is not a trust anchor for any end-entity (thus not a root).

MODEL II: STRICT HIERARCHY. A *strict CA hierarchy* is a system with multiple CAs organized as a tree with multiple levels of CAs, typically a closed system (single community). At the top is a single CA (depicted as the *root* of an inverted tree), followed by one or more levels of *intermediate CAs*; see Fig. 8.6(a). CAs at a given tree level issue

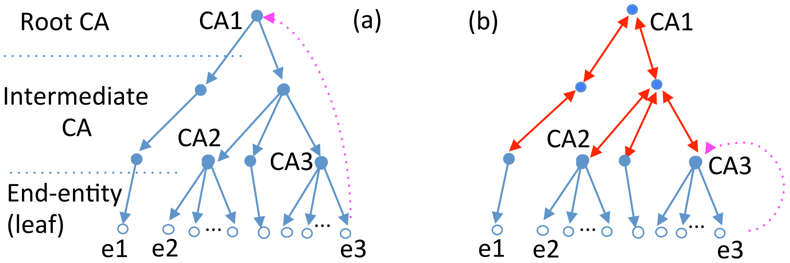


Figure 8.6: Model II, (a) Strict CA hierarchy trust model. (b) Hierarchy with reverse certificates. Nodes are certificates. Solid arrow points from certificate issuer to subject. Double-arrow means CA-certificates in both directions. Dotted arrow shows trust anchor.

certificates for the public keys of CAs at the next-lower level, until at a final *leaf-node* level, the keys in certificates are (non-CA) end-entity public keys. Typically, end-entities within the community have their software clients configured with the root CA public key as their trust anchor. A major advantage of a strict hierarchy is clearly defined trust chains starting from the root. In figures showing both trust anchors and certifications, the visual trust chain begins by following a dotted arrow from a leaf to a trust anchor, and then a path of solid arrows to another leaf. As a practice example, in Fig. 8.6(a), trace out the trust chain path from e3 to e2; then do so in Fig. 8.6(b).

HIERARCHY WITH REVERSE CERTIFICATES. A generalization of the strict hierarchy is a *hierarchy with reverse certificates*. The tightly structured hierarchical design is retained, with two major changes. 1) CAs issue certificates not only to their immediate children CAs in the hierarchy, but also to their parent (immediate superior); these *reverse certificates* go up the hierarchy. Figure 8.6(b) shows this by using double-ended arrows. 2) A leaf is given as trust anchor the CA that issued its certificate (not the root CA), i.e., its "local" CA, closest in the hierarchy. Trust chains therefore start at the local CA and progress up the hierarchy and back down as necessary.

MODEL III: RING-MESH OF TREE ROOTS. Returning to the base of multiple single-CA domains, suppose each single-CA domain is now a multi-CA system formed as a tree or hierarchy. The distinct trees are independent, initially with no trust cross-connects. Now, similar to connecting single-CA systems per Fig. 8.5, connect instead the root CA nodes of these trees. As before, topologies to consider include complete pairwise cross-connects, rings, and a bridge CA model. We collectively call these options a *ring-mesh of tree roots*. The end result is a system joining multiple hierarchical trees into a trust community by CA-certificates across subsets of their top CAs. If there are, say, 10 multi-CA trees, a fully-connected graph with all (10 choose 2) pairs of roots cross-certifying is possible, but in practice all such cross-certificate pairs might not be populated—e.g., not all 10 communities may wish to securely communicate with each other (indeed, some may not trust each other). If root CAs are equal peers, the bridge CA topology may be preferred. For Model III, the trust anchor configured into end-entities is often the root key of their original tree (and/or their local CA key); from this, trust chains including CA-certificates allow derived trust in the leaf nodes of other trees.

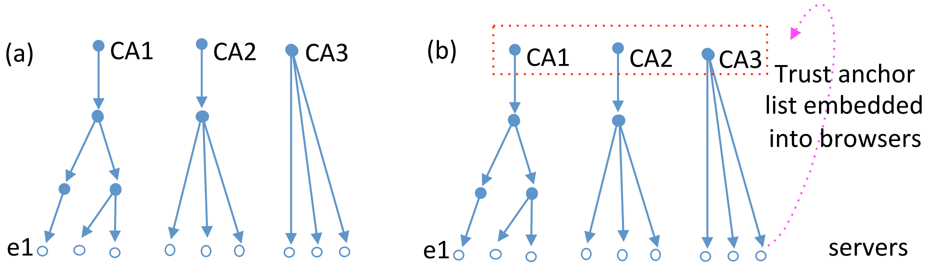


Figure 8.7: Model IV: Browser trust model. Built from multiple disjoint hierarchical trees (a), there are no CA cross-certificates in the browser trust model (b), and leaf certificates are for web servers (e.g., e1). Solid arrow points from certificate signer to subject.

Model IV: Forest of hierarchical trees (browser model). Starting as in Model III with a forest of (disjoint) hierarchical trees, an alternative to joining communities through cross-certificates is to use *trust anchor lists* (Fig. 8.7). End-entities get as trust anchors not just a single root key, but a (typically large) set of public keys corresponding to a collection of root CAs. Despite no cross-certificates across the tops of disjoint trees, an end-entity can derive trust in the leaf nodes of each tree for which it has a trust anchor. This is the approach of the *browser trust model* (Section 8.5), used not to authenticate end-users, but rather to allow users' browsers to recognize TLS-supporting servers; thus tree leaf nodes correspond to certificates of servers (domains).

Model V: Decentralized CA trust (Enterprise PKI model). This model, also called a *network PKI* or *mesh PKI* architecture, may be viewed as putting bottom-level CAs in control, i.e., the CAs that issue certificates to end-entities. The trust anchor configured into an end-entity is the public key of the CA that issued its certificate (based on reasoning that a leaf system has strongest trust affiliation with the CA "closest" to it in a trust graph). Variations allow importing—often under system control in an enterprise deployment—additional trust anchors or trust anchor lists. The model allows cross-certificates to link PKI components and facilitates peer relationships (Fig. 8.8), complete

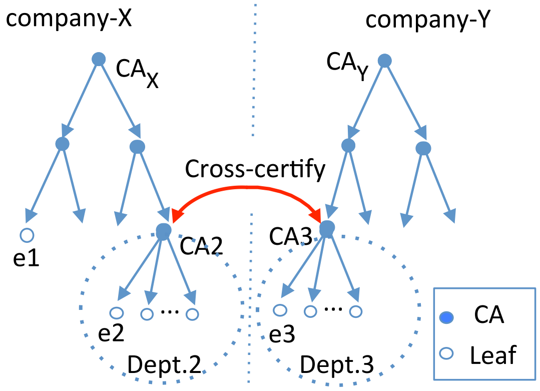


Figure 8.8: Model V: Enterprise PKI model. Cross-certifying peer departments lower in a hierarchy allows finer-grained trust peering than cross-certifying at the root.

networks, hierarchies, ring-meshes and bridge CAs. While not recommended, arbitrarily complex trust graphs are allowed—albeit complexity is limited to the CA network graph, whereas Model VI extends complexity further to include the end-user layer. The useful resulting architectures are those easily understood by administrators and users.

Motivation of Enterprise PKI model. In practice, PKIs are often built bottom-up, rather than fully planned before roll-out begins. Commercial products have focused on tools suitable for use within and between corporations or government departments, i.e., *enterprise products*. Companies may build a PKI first within a small department, then a larger division, then across international branches, and perhaps later wish to extend their community to allow secure communications with trusted partner companies. Practical trust models, and associated tools and architectures, accommodate this. A central idea is building (enlarging) *communities of trust*—keeping in mind that a vague definition of *trust* is unhelpful. In building a PKI and selecting a trust model, a helpful question to ask is: What is the PKI aiming to accomplish or deliver?

Example *(Decentralized model: cross-certifying subsidiaries).* An example of the enterprise PKI model is for *cross-certification of two subsidiaries*. Consider companies X and Y with their own strict hierarchies disjoint from each other (Figure 8.8). Each end-entity has as trust anchor the root CA of their own company. Suppose there is a desire for some entities in one company to recognize some certificates from the other. Adding root CA_Y of CompanyY as a trust anchor to end-entity e2 of CompanyX is a coarse-grained solution by which e2 will recognize all certificates from CompanyY; this could likewise be accomplished by having CA_X and CA_Y cross-certify, but in that case it would hold for all employees of each company (e.g., e1), not just e2. As a finer-grained alternative, suppose the motivation stems from e2 being in a division Dept2 that has need for frequent secure communication with Dept3 of CompanyY (as peer accounting departments). If these divisions have their own CAs, CA2 and CA3, those CAs could cross-certify as peer departments lower in the hierarchy. This allows e2 and e3 to trust each other's certificates via CA2-CA3 cross-certificates. Does e1 have a trust path to e3? Yes if end-entities have their own tree's root key as a trust anchor; no if end-entities only have their local CA keys as trust anchors. CompanyX can use extension fields (Section 8.2) in the certificate CA2 issues for CA3, to impose name, pathlength, and policy constraints (perhaps limiting key usage to email, ruling out VPN) to limit the ability of CompanyY's CA3 to issue certificates that CompanyX would recognize.

Example *(Decentralized model: single enterprise).* Consider a single, large corporation with a deep multi-CA strict hierarchy with reverse certificates (each division has its own CA). End-entities are configured with their local CA as trust anchor. Trust chains between all pairs of end-entities will exist, but adding direct cross-certificates between two divisions that communicate regularly results in shorter (simpler) chains.

Model VI: User-control trust model (web of trust). This model has no formal CAs. Each end-user is fully responsible for all trust decisions, including acting as their own CA (signing their own certificates and distributing them), and making individual, personal decisions on which trust anchors (other users' certificates or public keys) to import as trusted. The resulting trust graphs are ad hoc graphs connecting end-entities. This is the PGP model, proposed circa 1995 for secure email among small groups of technically oriented users, as discussed further in Section 8.6.

CA-CERTIFICATES VS. TRUST ANCHOR LISTS. To conclude this section, we note that it has highlighted two aspects that distinguish PKI trust architectures:

1. the trust anchors that end-entities are configured with (e.g., the public keys of CAs atop hierarchies, vs. local CAs); and
2. the relationships defined by CA-certificates (i.e., which CAs certify the public keys of which other CAs).

These aspects define how trust flows between trust domains (*communities of trust*), and thus between end-users.

8.5 TLS web site certificates and CA/browser trust model

The preceding sections discussed generic aspects of certificates and PKI. We now discuss their specific application to TLS—building on the browser trust model of Figure 8.7.

TRANSPORT LAYER SECURITY. *TLS* is the world's most widely deployed security protocol, and *de facto* standard means for securing web browser-server traffic. The 1994 predecessor of TLS, namely *SSL*, was designed for use with HTTP (to facilitate HTTPS), but also with the intent that almost any protocol using TCP could be easily modified to be run "over" TLS/SSL to add security. By 2000, SSL had already been used to add security to FTP (file transfer), SMTP (email), telnet (remote terminals) and LDAP (directory access). TLS/SSL had two original security goals: encryption of traffic between endpoints (confidentiality), and server authentication (through public-key certificates) to help assure that, e.g., a credit card number went to an intended server. From the outset, TLS also supported certificate-based client authentication (but this remains little used).

"TRUSTED" CERTIFICATE. The term *trusted certificate* means that a browser, or other client as a relying party, carries out certificate validation checks (Section 8.2) and concludes that from its viewpoint, the certificate is valid. A certificate trusted by one relying party might not be trusted by another (and might not be trustworthy).

GRADES OF TLS CERTIFICATES. As discussed (Section 8.1), before issuing a certificate, a CA should demand proof of knowledge of the requesting party's private key, and for TLS, test control or ownership of a domain or domain name. Whether and how a CA confirms the natural-world name of an organization results in three quality-related grades of TLS server certificates as follows, in increasing order of CA due diligence.

DV CERTIFICATES. For *Domain Validated* (DV) certificates, the `Subject` is validated minimally by demonstration of administrative control of the domain, e.g., the ability to respond to an email sent to a designated domain account (such as `admin@domain.com`), or the ability to publish a CA-specified string in a DNS domain record. No assurance is provided that the requesting party is associated with any real-world entity. DV certificates are inexpensive (e.g., $10–$50, or even free), and their issuance is often fully automated.

OV CERTIFICATES. Obtaining an *Organization Validated* (OV) certificate requires demonstration of domain control plus the requesting party passing further manual checks (e.g., confirmation of a claimed street address and organization business name), depending on a CA's *Certificate Practice Statement* (CPS) specifying the policy it operates under. The CA may cross-check a claimed business name with a commercial database (e.g., Dun & Bradstreet's list of 300 million businesses) and call-back to a publicly listed phone number from such a database. Browsers relying on OV certificates may display organization-related information to users who seek certificate-related information. Software can extract information from the certificate `Subject` field, whose populated subfields may include as noted earlier, e.g., Common Name, Organizational Unit, Organization, and Country. Most end-users notice no difference between DV and OV certificates (any differences in browser cues may be too obscure or not understood, as discussed further in Section 9.8).

EV CERTIFICATES. *Extended Validation* (EV) certificates were motivated by loss of confidence after low-assurance DV certificates emerged—for example, browser-trusted DV certificates from phishing sites result in the same visual assurances to end-users as do DV certificates from non-phishing sites, e.g., a TLS closed-lock icon in browser URL bars. Before issuing an EV certificate, the issuing CA is expected to manually verify:

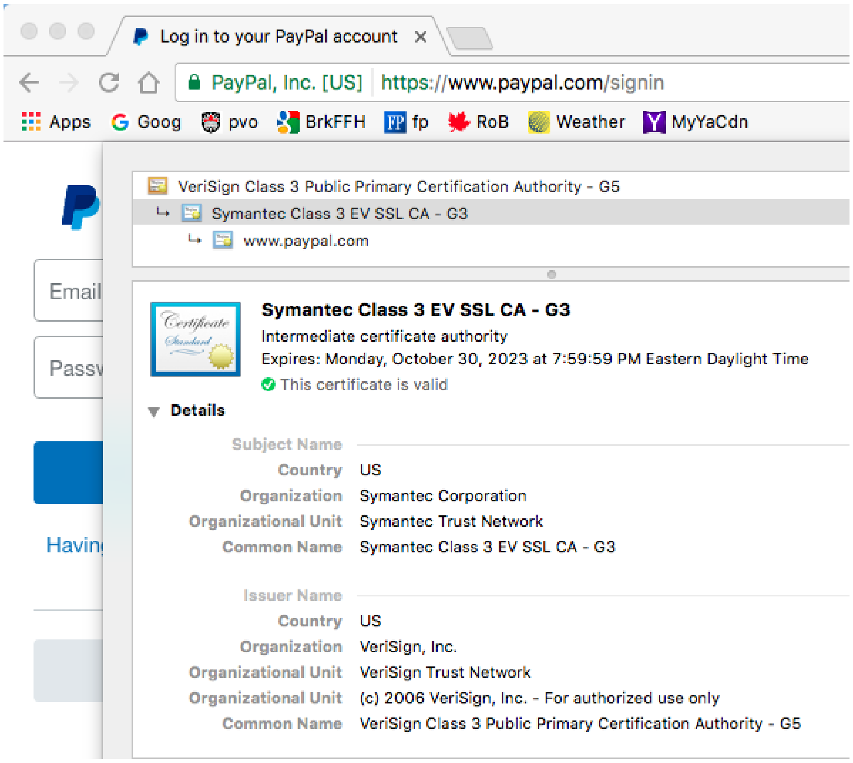
1. the real-world existence of the legal entity named as `Subject`;
2. the registration of an associated organization in government-recognized databases;
3. a physical, operational existence matching the location indicated in the certificate;
4. the requesting individual's identity and their authority to represent the organization;
5. exclusive control of the specified domain name.

The `Subject` DN must be fully qualified (wildcard domains like `*.google.com` are disallowed in EV certificates). An X.509v3 extension field `Certificate-Policies` is used, and includes a CA-specific policy *object identifier* (OID) and a URL pointing to a *Certificate Practice Statement*; the policy requires "timely" responses to browser revocation checks, among other things. The CA registers the OID with browser vendors as its EV identifier. On receiving an EV certificate from a visited site, the browser checks that the EV OID therein matches that pre-registered by the CA.

IV CERTIFICATES. In addition to DV, OV and EV grades, *IV certificates* (Individual Validated) are those that a user has chosen to accept (e.g., self-signed certificates, Section 8.2) despite not being recognized by their client software's trust anchors.

‡**Exercise** (ACME DV certificates). To remove cost and complexity barriers related to acquiring TLS server certificates, the *Let's Encrypt* service provides free automated web site certificates using ACME, a standardized certificate issuance protocol. a) Describe the roles of the following ACME components: accounts, orders, authorizations, challenges. b) Describe three types of challenges suitable to verify control of a domain. (Hint: [34].)

‡CA/BROWSER FORUM AND EV CERTIFICATES. The *CA/Browser Forum* is a voluntary industry association of CAs and browser vendors with joint interests in TLS certificates. It governs operational practices for issuing EV certificates, and also publishes guidance for issuing and managing non-EV certificates. EV guidelines require that CAs publish operational policies; conformance is third-party audited. However, without better


Log in to your PayPal account ×
PayPal, Inc. [US] https://www.paypal.com/signin
Apps
Goog
pvo
BrkFFH
fp
RoB
Weather
MyYaCdn
Email
Passw
Havin

VeriSign Class 3 Public Primary Certification Authority - G5
↳ Symantec Class 3 EV SSL CA - G3
↳ www.paypal.com

Symantec Class 3 EV SSL CA - G3
Intermediate certificate authority
Expires: Monday, October 30, 2023 at 7:59:59 PM Eastern Daylight Time
✓ This certificate is valid

▼ **Details**

Subject Name	
Country	US
Organization	Symantec Corporation
Organizational Unit	Symantec Trust Network
Common Name	Symantec Class 3 EV SSL CA - G3
Issuer Name	
Country	US
Organization	VeriSign, Inc.
Organizational Unit	VeriSign Trust Network
Organizational Unit	(c) 2006 VeriSign, Inc. - For authorized use only
Common Name	VeriSign Class 3 Public Primary Certification Authority - G5

Figure 8.9: Intermediate CA EV certificate (Chrome 56.0.2924.87 browser). The URL bar displays organization details “Paypal, Inc. [US]” of the server site between the lock icon and “https:”. The grey-highlighted “Symantec Class 3 EV SSL CA - G3” shows this as the middle of three certificates in a chain; further detail notes this is an “Intermediate certificate authority”. When imaged, these details were viewed from the pulldown menu View→Developer→DeveloperTools→[View Certificate→Details].

means in browsers to effectively communicate the differences between EV and (OV, DV) certificates, it remains unclear what added value EV certificates deliver to users. This is discussed further in Section 9.8, along with the challenge of conveying to users the presence of EV certificates. For example (Fig. 8.9) on browser user interfaces, EV certificates may result in the URL bar/lock icon being colored differently (this varies by browser, and over time) and the URL bar displaying a certificate `Subject`’s name and country.

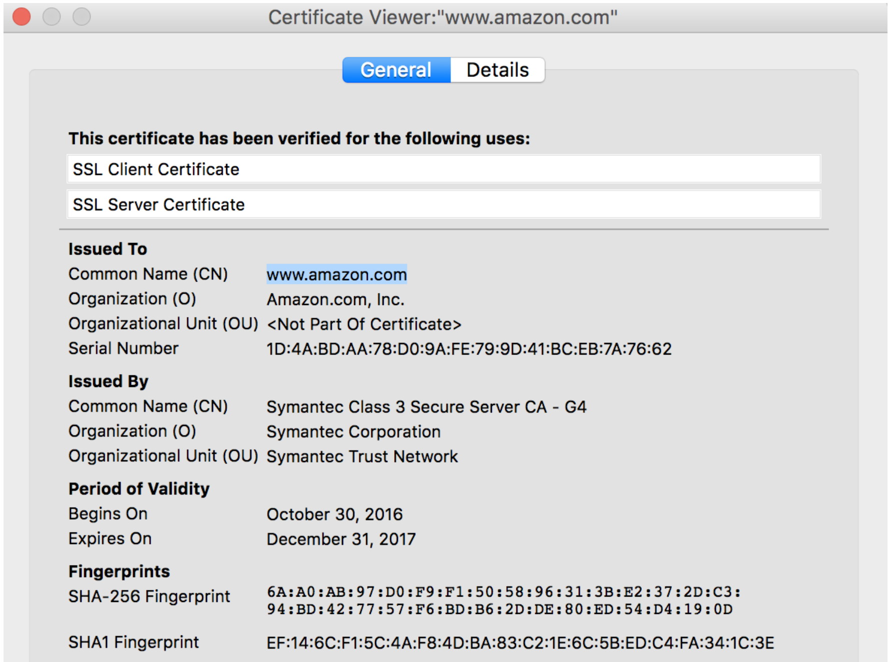
SELF-SIGNED TLS SERVER CERTIFICATES. Self-signed TLS certificates were once common (before free DV certificates became popular); non-commercial sites often preferred to avoid third-party CAs and related costs. Over time, browser dialogues were reworded to discourage or entirely disallow this (recall Fig. 8.3, page 219). Relying on self-signed certificates (and/or blind TOFU) should be strongly discouraged for non-leaf certificates, due to trust implications (private keys corresponding to CA certificates can sign new certificates). However, this is one method for distributing email leaf certificates (incoming email offers a sender’s encryption and signature public keys).

Example *(Number of TLS CAs).* A March 2013 Internet study observed 1832 browser-trusted CA signing certificates, including both trust-anchor CA and intermediate-CA certificates, associated with 683 organizations across 57 countries [14].

Exercise (Self-signed certs). Build your own self-signed certificate using a popular crypto toolkit (e.g., OpenSSL). Display it using a related certificate display tool.

Exercise (Domain mismatch). Discuss practical challenges related to *domain mismatch errors*, i.e., checking that the domain a browser is visiting via TLS matches a suitable subfield of a certificate `Subject` or `Subject-Alternate-Name` (hint: [51]).

Exercise (CA compromises). Look up and summarize the details related to prominent compromises of real-world CAs (hint: [3]).



This certificate has been verified for the following uses:

SSL Client Certificate

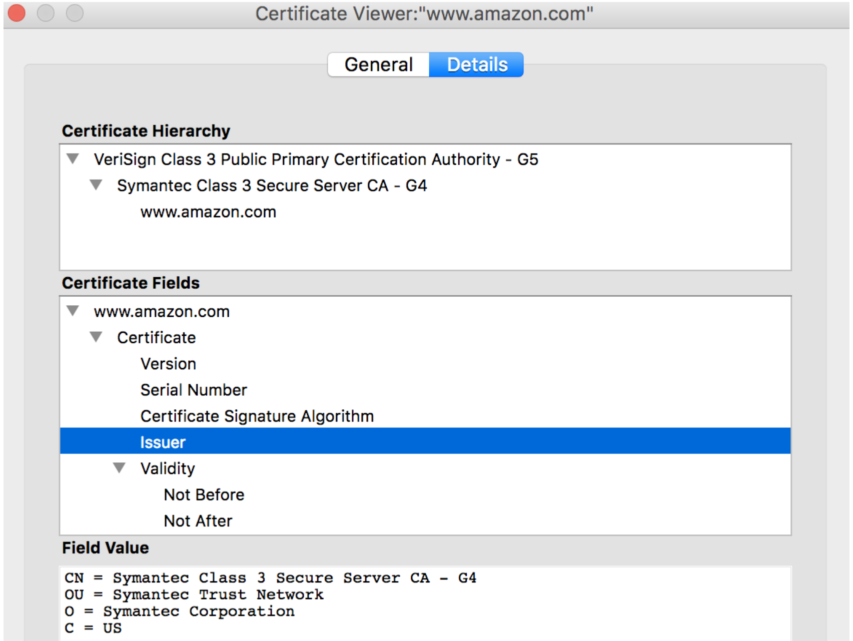
SSL Server Certificate

Issued To	
Common Name (CN)	www.amazon.com
Organization (O)	Amazon.com, Inc.
Organizational Unit (OU)	<Not Part Of Certificate>
Serial Number	1D:4A:BD:AA:78:D0:9A:FE:79:9D:41:BC:EB:7A:76:62
Issued By	
Common Name (CN)	Symantec Class 3 Secure Server CA - G4
Organization (O)	Symantec Corporation
Organizational Unit (OU)	Symantec Trust Network
Period of Validity	
Begins On	October 30, 2016
Expires On	December 31, 2017
Fingerprints	
SHA-256 Fingerprint	6A:A0:AB:97:D0:F9:F1:50:58:96:31:3B:E2:37:2D:C3: 94:BD:42:77:57:F6:BD:B6:2D:DE:80:ED:54:D4:19:0D
SHA1 Fingerprint	EF:14:6C:F1:5C:4A:F8:4D:BA:83:C2:1E:6C:5B:ED:C4:FA:34:1C:3E

Figure 8.10: General tab, TLS site certificate (Firefox 55.0.1 UI). The UI tool, under **Issued To**, displays the certificate `Subject` with subfields CN (giving domain name `www.amazon.com`), O and OU; compare to Figure 8.11. **Issued By** indicates the certificate-signing CA. Under **Fingerprints** are the hexadecimal values of the certificate hashed using algorithms SHA-256 and SHA1, to facilitate a manual security cross-check.

BROWSER INTERFACE DIALOGUES ON CERTIFICATES. Information related to TLS certificates can be found, e.g., by clicking on the closed-lock icon in a browser URL bar. Figure 8.3 shows a warning dialogue on encountering an (untrusted) self-signed site certificate. Figure 8.10 gives high-level information about a site certificate. Figure 8.11 gives detailed information; the interface allows exploration of the certificate chain.

BROWSER TRUST MODEL ISSUES. It is well recognized that the traditional browser trust model has major vulnerabilities—many due to a base design flaw that allows any CA to issue a certificate for any domain. This gives all trust anchors equal power to commit



Certificate Hierarchy

- VeriSign Class 3 Public Primary Certification Authority - G5
  - Symantec Class 3 Secure Server CA - G4
    - www.amazon.com

Certificate Fields

- www.amazon.com
  - Certificate
    - Version
    - Serial Number
    - Certificate Signature Algorithm
    - Issuer
    - Validity
      - Not Before
      - Not After

Field Value

```
CN = Symantec Class 3 Secure Server CA - G4
OU = Symantec Trust Network
O = Symantec Corporation
C = US
```

Figure 8.11: Details tab, TLS site certificate (Firefox 55.0.1). The **Certificate Hierarchy** segment displays the certificate chain. As the user scrolls through the middle portion to access additional fields, a selected field ("Issuer" here) is highlighted by the display tool and its value is displayed in the lower portion. Notations OU (organizational unit), O (organization), and C (country) are remnants of X.500 naming conventions.

a browser to trust any web site. We define a *rogue certificate* as one created fraudulently, not authorized by the named `Subject` (e.g., created by an untrustworthy CA or using the private key of a compromised CA). A list of main limitations follows.

1. Rogue certificates are accepted (sometimes called *certificate substitution* attacks). They are deemed valid by all browsers housing a corresponding CA public key as a trust anchor. The trust model is thus *fragile*, in that regardless of the strength of other CAs, the entire system can be undermined by a single rogue CA (weak link) that gains the endorsement of a trust anchor CA. This violates principle P13 (DEFENSE-IN-DEPTH).

2. *TLS-stripping attacks* are easily mounted. Here a legitimate server signal to a browser to upgrade from HTTP to HTTPS, is interfered with such that no upgrade occurs. Data transfer continues over HTTP, without cryptographic protection. One solution is to eliminate HTTP entirely, mandating HTTPS with all sites; this suggestion is not popular with sites that do not support HTTPS. A related option is mechanisms that force use of HTTPS whenever a browser visits a site that supports it; a browser extension pursuing this option is aptly called *HTTPS Everywhere*. Vulnerability to TLS stripping may be viewed as breaking P2 (SAFE-DEFAULTS), as the current default is (unsecured) HTTP.

3. Revocation remains poorly supported by browsers. When revocation services are unavailable, browsers commonly proceed as if revocation checks succeeded. Such "fail-

ing open" contradicts P2 (above), and also violates P4 (COMPLETE-MEDIATION).

4. *Trust agility* is poorly supported. This refers to the ability of users to alter trust anchors. Most users actively rely on few trust anchors; browser and OS vendors commonly embed hundreds. This violates principle P6 (LEAST-PRIVILEGE) as well as principle P17 (TRUST-ANCHOR-JUSTIFICATION), and is particularly dangerous as certificate chaining then transitively extends (false) trust in one trust anchor to many certificates.

5. Intermediate CAs are unaccountable. An intermediate CA issued a certificate by a rogue CA can also create rogue site certificates. Detecting rogue certificates is complicated by intermediate CAs being largely invisible to users. The resulting lack of accountability (P14, EVIDENCE-PRODUCTION) enables *compelled certificate* attacks, whereby a CA is coerced, by shady organizations governing or regulating them, to issue intermediate CA certificates to facilitate surveillance via middle-person attacks.

As a case study of flaws in a system in use for 25 years, these issues remain useful to understand, even should they be resolved by future alterations of the browser trust model.

‡**Exercise** (Public log of TLS certificates). *Certificate Transparency* (CT) is among promising proposals to address limitations of the CA/browser trust model. The high-level idea is to require that all certificates used in TLS be published in a publicly verifiable log. a) Summarize its technical design, and advantages over mainstream alternatives (hint: [31]). b) Summarize the findings of a deployment study of CT (hint: [48]). c) Summarize the abstract technical properties CT aims to deliver (hint: [13]).

‡**Exercise** (DANE certificates). As an alternative to CA-based TLS certificates, certificates for TLS sites (and other entities) can be distributed by association with DNS records and the *DANE* protocol: DNS-based Authentication of Named Entities. Describe how DANE works, and its relationship to DNSSEC (hint: [22]).

‡**Exercise** (Heartbleed incident). Standards and software libraries allow concentration of security expertise on critical components of a software ecosystem. This also, however, concentrates risks. As a prominent example, the *Heartbleed* incident arose due to a simple, but serious, implementation flaw in the OpenSSL crypto library. Give a technical summary of the OpenSSL flaw and the Heartbleed incident itself (hint: [15]).

‡**Exercise** (CDNs, web hosting, and TLS). *Content delivery networks* (CDNs), involving networks of proxy servers, are used to improve performance and scalability in delivering web site content to users. They can also help mitigate *distributed denial of service* (DDoS) attacks through hardware redundancy, load balancing, and isolating target sites from attacks. When CDNs are used to deliver content over HTTPS, interesting issues arise, and likewise when web hosting providers contracted by web sites must deliver content over HTTPS. Explore and report on these issues, including unexpected *private-key sharing* and use of *cruiseliner certificates* (hint: [32, 8]).

‡**Exercise** (TLS challenges in smartphone and non-browser software). Discuss certificate validation challenges (and related middle-person attacks) in use of TLS/SSL by: (a) smartphone application software (hint: [16]); and (b) non-browser software (hint: [20]).

8.6 Secure email overview and public-key distribution

In *end-to-end secure email*, messages are encrypted and digitally signed on the originating user's device, with verification and decryption at the recipient's device. Among longstanding barriers to its wide deployment are key management of one's own certificates and private keys, and acquisition of trustworthy public keys of others. Secure email played a large role in developing PKI and certificate management technology and standards (even before TLS existed). Three technologies have stood out (Section 8.7 gives details):

1. PEM (Privacy-Enhanced Mail). With a one-root PKI hierarchy, PEM is of historic and technical interest as the first comprehensive secure email effort (circa 1990-1995).
2. PGP (Pretty Good Privacy). With ad hoc trust management for highly technical users, PGP's 1991 design suits private individuals and relatively small, static groups.
3. S/MIME. Secure MIME, when used with centralized certificate management, is suitable for enterprise users, e.g., government departments and corporations.

This section provides a generic overview of secure email.

EMAIL TRANSPORT. A *mail user agent* (MUA) transfers mail to the originating user's sending mail server or mail submission agent (MSA). The domain from a recipient's email address is used to locate, via DNS, the recipient's *mail delivery agent* (MDA). The message is transferred to this MDA via one or more *mail transfer agents* (MTAs) using the *simple mail transfer protocol* (SMTP). The recipient's MUA (mail client) retrieves the message from this MDA by a message retrieval protocol—either a proprietary or a standard protocol such as IMAP (often used to manage mail from multiple devices) or POP3 (which supports deleting server-based copies after retrieval). These protocols allow mail storage on servers, clients, or both—thus end-to-end secure email designs must consider whether to store delivered email in plaintext, or encrypted (and if so, whether to re-encrypt by means independent of mail transport encryption).

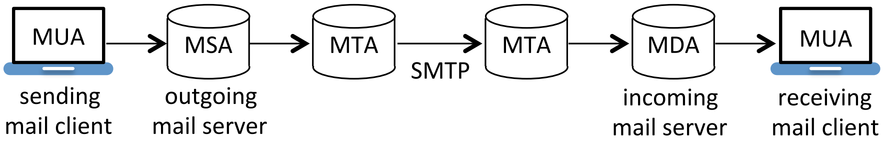


Figure 8.12: Mail transfer model. The MSA may be combined with MTA functionality.

MESSAGE STRUCTURE: REGULAR EMAIL. Mail transport standards define message *envelope* and *content* sections. How mail clients (MUAs) display mail to users is a separate matter. The envelope contains fields used during transmission, e.g., message transport details, timestamps from MTAs, and path-related information. The content section—what the user requests be delivered—includes a *header* section (with fields such as `From`, `To/CC`, `Date`, `Subject`) and a message *body*, separated by a blank line.

MESSAGE STRUCTURE: SECURE EMAIL. A main goal in designing a message structure supporting end-to-end secure email is backwards compatibility and interoperability with existing transport systems and mail clients. To this end, security-specific functionality is typically restricted to the content section—inserting, for example, a new

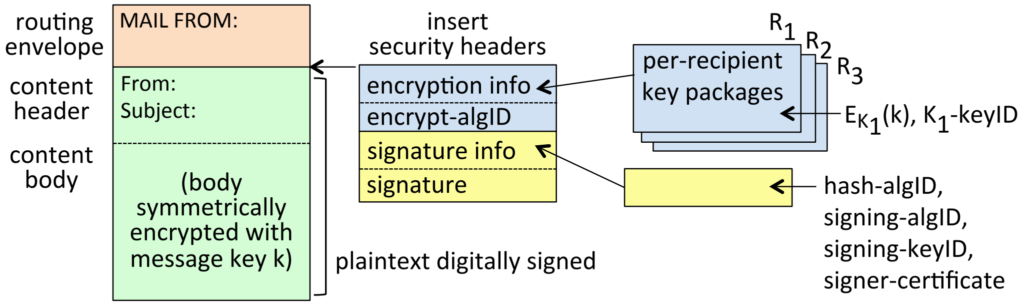


Figure 8.13: Schematic of secure email message (conceptual). Security headers allow decryption and signature verification. K_1-keyID identifies the first recipient's certificate or public key, e.g., via the serial number of a certificate issued by a specified CA. The key transport case is shown; per-recipient key packages may include additional keying material from the sender, in the case of (e.g., Diffie-Hellman) key agreement.

interior *security header* section providing meta-data to support signature verification and mail decryption. If encoded suitably (i.e., using printable characters), legacy (plaintext-only) clients can then display the interior security header and encrypted body as labeled fields followed by meaningless, but printable, ASCII characters. Commonly, the sending client uses a symmetric key k (*message key*) to encrypt the plaintext body. The plaintext body (plus content header) is also hashed and digitally signed. The security header includes fields providing (Fig. 8.13):

- for each recipient R_i, a copy of k encrypted under R_i's public key K_i, plus data identifying K_i (for R_i's client to find its package, and identify its decryption private key);
- an identifier for the symmetric encryption algorithm used, plus any parameters;
- the sender's digital signature, plus an identifier of the signing algorithm; and
- an identifier of the sender's public key to verify the signature (optionally also, a certificate containing the public key, and/or a chain of certificates).

It is common to include a copy of k encrypted under the sender's own K_i, allowing senders to decrypt stored copies of sent messages. While many key management issues here mirror those in TLS, differences arise due to email's *store-and-forward* nature (vs. real-time TLS); one challenge is acquiring a recipient encryption public key for the first encrypted mail sent to that party. We next consider two options for *public-key distribution*, i.e., distributing (acquiring) public keys of the intended recipient (encryption public key) and sender (signature verification public key). Note that a relying party's trust in such a key differs from their possession of it, and is enabled by different PKI trust models.

CENTRALIZED PUBLIC-KEY DISTRIBUTION. Whereas Chapter 4 discussed key distribution using symmetric-key techniques, here we mention two methods for distribution of public keys (as in Chapter 4, a centralized model avoids the "n^2 key distribution" issue). In typical security applications using public keys, each of n end-parties has at least one public-private key pair, to facilitate authentication (signatures) and/or key establishment with other parties, e.g., to set up session keys or, in our present application,

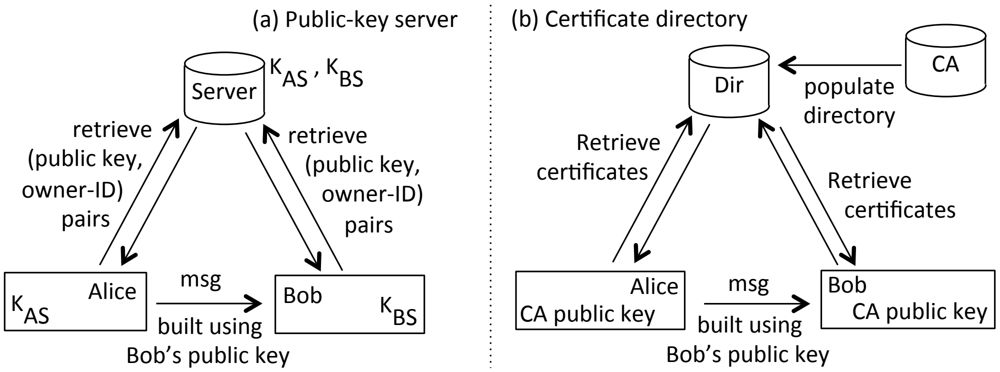


Figure 8.14: Centralized distribution of public keys. (a) Public-key server. Pairs (key, ownerID) are retrieved in real time from a trusted server, secured by session-key MACs; see related discussion of Method V on page 223. (b) Certificate directory. Signatures on certificates replace the need for a trusted channel.

per-message email keys. The first option for acquiring the public key of another party uses an online trusted *public-key server* (Fig. 8.14a). End-parties retrieve from it in real time, immediately before each communication session, <public key, ownerID> pairs integrity-protected by a session key shared between server and end-party. The server is like a KDC (Chapter 4), but now distributes public keys.

CERTIFICATE DIRECTORY. The second option involves a repository (*certificate directory*) of CA-signed certificates (Fig. 8.14b). Public keys may now be retrieved at any time; each party acquires from the CA a certificate for its own public key(s) at registration. Certificates are made available to other end-parties by a subject directly, or via the directory. Directories themselves need not be trusted, as trust in the public keys delivered stems from the verification of signatures on certificates; corresponding private keys are held by end-parties, not the directory or CA. An end-party that is to rely on the public key in a certificate requires an authentic copy of the public key of the CA that signed the certificate, or a certificate chain connecting the certificate to a trust anchor. Returning to our email application, an email sender needs, as initial material to encrypt email for a recipient, the recipient's encryption public key. For such store-and-forward protocols, this public key can be obtained from the directory, or by earlier email or out-of-band means (for real-time communications protocols, a certificate can be delivered in-protocol). Since a signature verification public key is not required until a recipient receives email, a certificate providing this public key can be sent with the email itself.

‡PROS AND CONS OF CERTIFICATES. Use of certificates may facilitate audits of all public keys ever associated with an end-party, should anyone question server trustworthiness. As a disadvantage, using a certificate some period after creation raises the issue of whether its public key remains valid when used, thus requiring certificate revocation infrastructure (Section 8.3). Certificate validation also requires (Section 8.2) checking that a certificate's `Subject` maps to an intended entity; this can be tested by software if a precise domain name (e.g., from the URL bar) or email address is known, but, e.g., a mail client

can make no decision given only an asserted ID `name@domain.com`, if the user is unsure of the address; an analogous issue exists with key servers (cf. ownerID, Fig. 8.14a).

Exercise (Cleartext header section). A typical end-to-end secure email design (Fig. 8.13) leaves the content header unencrypted. What information does this leave exposed to eavesdroppers? What are the obstacles to encrypting the content header section?

Exercise (Order of signing and encrypting). Commercial mail products may first compute a digital signature, and then encrypt both the signature and content body. What advantage does this offer, over first encrypting the body and signing afterwards?

‡FURTHER CONTEXT. In contrast to end-to-end secure email, common browser-based mail clients (*webmail interfaces*) use TLS link encryption between users and mail servers, but the message body is then available as cleartext at various servers. End-to-end secure email deployment is complicated by *mail lists* and *mail forwarding*; these are beyond our scope, as is *origin-domain authentication* used by mail service providers.

‡END-TO-END ENCRYPTION VS. CONTENT SCANNING. Various measures are used by mail service providers to combat spam, phishing, malicious attachments (including executables that users may invoke by double-clicking), and embedded malicious scripts (which some MUAs that support HTML email automatically execute). As end-to-end encryption renders plaintext content inaccessible to mail-processing servers, this precludes content-based malware- and spam-detection by service providers. While *key escrow* architectures can provide plaintext access at gateway servers, e.g., by retrieving an escrowed copy of the mail originator's decryption private key, costs include performance, defeating end-to-end encryption, and risks due to added complexity and attack surface.

8.7 ‡Secure email: specific technologies

We now discuss three end-to-end secure email technologies: S/MIME, PEM, and PGP.

S/MIME. Secure Multipurpose Internet Mail Extensions (S/MIME) is a suite of standards for end-to-end secure mail compatible with existing mail transport protocols. It uses X.509v3 certificates and a centralized trust model, with trust in public keys determined by CA signatures on certificates and the trust anchors configured in (or used by) mail clients. Enterprise deployments typically rely on *certificate directories* (Fig. 8.14b), e.g., using LDAP as the access protocol. Mail clients are relied on, as usual, to map recipient names to email addresses; LDAP queries then return certificates. CAs are relied on to make revocation information available for certificates they issue. Mail can only be encrypted for recipients having encryption public keys, and when these are available to senders. *Signed-only email* can be read by regular mail clients that support MIME—the S/MIME *detached signatures* mode conveys signature data in a separate MIME part of a multipart/signed message. Most mail products targeting enterprise markets support S/MIME.

SECURE EMAIL IN CLOSED COMMUNITIES. S/MIME has been successfully deployed in closed communities (e.g., large corporations and governments). Internal staff may dictate the mail clients used (supporting S/MIME and management of user certificates and private keys), help employees acquire and install certificates, and configure clients with trust anchors matching enterprise policy, and with access to suitable certificate directories. Enterprise PKI trust models (Section 8.4) facilitate trust with similarly configured peer organizations. This leaves unaddressed secure communication with users beyond the closed community. Making public keys available, e.g., by inclusion in preceding cleartext email, does not resolve whether keys can be trusted—that depends on CA/PKI models and trust anchors. In contrast, open communities have users with widely varying requirements, and no small set of CAs is naturally trusted by all; thus a one-size-fits-all solution is elusive. One option is to continue with plaintext email. A second is to migrate outsiders into the closed community—but by definition, a closed community does not contain everyone. A third option is ad hoc trust management (PGP, below).

PEM (PRIVACY-ENHANCED MAIL). The first major secure email effort began in 1985. PEM used X.509 certificates and a hierarchy with one root, the Internet PCA Registration Authority (IPRA), issuing certificates starting all certificate chains. The IPRA public key was embedded in all PEM mail clients. Below this root CA at hierarchy level two, Policy CAs (PCAs) operating under designated policies issued certificates to intermediate CAs or directly to end-users—e.g., high-level assurance PCAs (for enterprise users), mid-level assurance PCAs (for educational users), residential PCAs (for private individuals), and *persona* PCAs (for anonymous users). PEM clients were trusted to retrieve—from local caches or directories—and verify user certificates corresponding to email addresses. A CRL-typed mail message delivered CRLs, with PCAs responsible for revocation information being available. Subject distinguished names (DNs) followed the CA hierarchy (i.e., DNs were subordinate to the issuing CA's name), restricting the *name space* for which each CA was allowed to issue certificates. PEM was superseded by S/MIME.

PGP: CONTEXT. Released as open-source file encryption software in 1991, PGP's primary use is for end-to-end secure email. It was motivated by a desire to empower individuals in opposition to centralized control, and against the backdrop of (old) U.S. crypto export controls. Its complicated evolution has included intentional message format incompatibilities (driven by patent license terms), algorithm changes to avoid patents, corporate versions, IETF-standardized OpenPGP, and later implementations (e.g., Gnu Privacy Guard/GPG). Despite confusion on what "PGP" means (e.g., a message format, format of public keys, trust model, company), and recent PGP implementations pursuing interoperability with X.509 certificates, its core concepts remain an interesting case study.

PGP: CORE CONCEPTS. Core PGP avoids CAs and X.509 certificates. Instead it uses a *PGP key-packet* (bare public key), which, when associated by client software to a userID (username and email address), is a *lightweight certificate* (unsigned). A collection of one or more keys is a *keyring*. A *public keyring* holds public keys; a *private keyring* holds a user's own private keys, individually encrypted under a key derived from a user-chosen *passphrase*. PGP's preferred method for one user to trust that a public key belongs to another is an in-person exchange of keys (originally by floppy disk); the user then has their client software tag the key-packet as trusted. Publishing a hexadecimal hash string corresponding to a PGP public key on a business card or web site, or relaying this by phone, would facilitate cross-checking. As this scales poorly, *trusted introducers* were added: if Alice designates Tom as a trusted introducer, and Tom endorses Bob's key-packet, Alice's client will trust Bob's key-packet also. Users configure their client to designate trusted introducers as fully or *partially trusted*; e.g., a key-package, to be client-trusted, must be endorsed by one fully trusted or two partially trusted introducers. Trusted introducers thus serve as informal end-user CAs. The PGP *web of trust* results.

PGP TRANSFERABLE KEYS. To help client software manage PGP key-packets (bare keys), they are accompanied by further fields creating *transferable public keys*. The bare key is followed by one or more *UserID packets* each followed by zero or more *signature packets* (endorsements attesting the signer's belief that the public key belongs to the UserID). Thus transferable public keys reconstruct the basic idea of X.509 certificates, replacing the signature of a centralized CA with possibly multiple endorsements of various end-users. Users are encouraged to upload transferable public keys to *PGP keyservers* hosting public keyrings of such keys; the trust placed in such keys by others depends on how PGP clients of downloading users are locally configured to evaluate endorsements.

PGP ISSUES. PGP's core architectural design reflects its original objectives, but is not expected to match secure email requirements in general. Challenges include these:

1. The manual exchange of public keys, and *ad hoc* web of trust, do not scale to larger communities. (Ironically, as an initial deployment advantage, a small closed group can get started with manual key distribution without needing to first set up a heavyweight infrastructure.)
2. User management of trust requires technical expertise that ordinary users lack, including the ability to distinguish between trusting a key for personal use, endorsing keys for other users, and designating trusted introducers in PGP clients.
3. The non-centralized model leaves revocation of PGP keys unresolved. Users are responsible for communicating key revocation to all others possibly relying on their key (including through trusted introducers), yet there appears no reliable means to do so.
4. Poor usability, in part due to lack of seamless integration into popular email clients, has impeded mainstream acceptance and deployment of PGP functionality.

SECURE EMAIL STATUS IN PRACTICE. Email continues to be a dominant communication tool, despite ubiquitous use of popular messaging applications, and older text-messaging technology. End-to-end secure email, however, enjoys comparatively little public deployment, due to multiple factors. Competing email technologies result in interoperability and deployment problems. Certificate and key management tools fall short on both usability and availability, particularly in open communities lacking enterprise expertise and administration. Stalemates appear unresolvable between stakeholders with incompatible priorities—e.g., those of law enforcement vs. privacy enthusiasts, and traditional end-to-end encryption at odds with email service providers' desire for access to message content for malware and spam filtering. Adoption of webmail services (vs. older client-based mail) is another complication. While it appears unlikely that all barriers to wide use of end-to-end secure email will disappear, its history remains among the most interesting case studies of real-world adoption of secure communication technologies.

8.8 ‡End notes and further reading

For authoritative treatments of X.509 certificate-related security, PKI architecture and trust models, see Housley [24] (including the U.S. Federal Bridge CA project) and Adams [2]; see also Kaufman [27] and Menezes [35, Chapter 13]. RFC 5280 [10] specifies Internet profiles for X.509v3 certificates and CRL mechanisms. Baseline standards ITU-T X.509:2000 and ISO/IEC 9594-8:2001 specify that the *name constraints* extension should be marked critical; RFC 5280 mandates this. For OCSP, see RFC 6960 [44]; for the PKIX Certificate Management Protocol (CMP), see RFC 4210 [1]. Additional mechanisms support certificate revocation: *indirect CRLs*, *redirect CRLs*, *certificate revocation trees* (CRTs), and others [36, 2]. For *EV certificate* guidelines, see documents from the CA/Browser forum [5, 6]. Figure 8.4's revocation timeline is based on Just [26]. For the efficacy of various textual and graphical representations for comparing key fingerprints, see Tan [49]. For analysis of using leap-of-faith (TOFU) including in SSH, see Pham [41]. Related to TOFU, for *key continuity management* (in S/MIME), see Garfinkel [19].

For TLS/SSL history, see Rescorla [42]. TLS 1.3 [43] was more redesign [40] than revision of TLS 1.2 [12]. Clark [9] summarizes challenges with HTTPS and its certificate trust model; Liu [33] measures web PKI support for revoking TLS certificates. Larisch [30] proposes a method for pushing such revocations to browsers. The TLS *Heartbleed* incident revealed an inability to handle massive-scale TLS certificate revocations [52, 15]. Saghoian [47] discusses *compelled certificates*. Liang [32] and Cangialosi [8] discuss how *content distribution networks* (CDNs) and site host providers interact with TLS. Kranch [29] explores addressing TLS stripping by HTTPS strict transport security (*HSTS*), and rogue certificates by *public key pinning*.

Orman [39] summarizes technical challenges in implementing S/MIME and PGP; see also Kaufman [27], including for S/MIME-based Lotus Notes and insights on PKI, and Zurko [55] on usability in Notes. Garfinkel [18] includes suggestions for bootstrapping key distribution in S/MIME. S/MIME uses the Cryptographic Message Syntax (CMS) of RFC 5652 [23]; for S/MIME v4.0 see RFCs 8550 and 8551 [45, 46] (cf. RFC 2634 [21]). For PEM, see Kent [28]. For PGP, the original definitive reference is Zimmermann [54]; see also Garfinkel [17], RFC 4880 [7] for OpenPGP message formats (also `https://www.openpgp.org/`), comments from Vaudenay [50, §12.4], and more recent views of proponents [53].

References

[1] C. Adams, S. Farrell, T. Kause, and T. Mononen. RFC 4210: Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP), Sept. 2005. Standards Track; obsoletes RFC 2510; updated by RFC 6712.

[2] C. Adams and S. Lloyd. *Understanding Public-Key Infrastructure (2nd edition)*. Addison-Wesley, 2002.

[3] A. Arnbak, H. Asghari, M. van Eeten, and N. V. Eijk. Security collapse in the HTTPS market. *Comm. ACM*, 57(10):47–55, 2014.

[4] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten. RFC 8555: Automatic Certificate Management Environment (ACME), Mar. 2019. Proposed Standard.

[5] CA/Browser Forum. Baseline requirements for the issuance and management of publicly-trusted certificates. Version 1.5.6, 5 February 2018. `https://cabforum.org`.

[6] CA/Browser Forum. Guidelines for the issuance and management of Extended Validation certificates. Version 1.6.8, 21 December 2017 (effective 9 March 2018). `https://cabforum.org`.

[7] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. RFC 4880: OpenPGP Message Format, Nov. 2007. Proposed Standard; obsoletes RFC 1991, RFC 2440.

[8] F. Cangialosi, T. Chung, D. R. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem. In *ACM Comp. & Comm. Security (CCS)*, pages 628–640, 2016.

[9] J. Clark and P. C. van Oorschot. SoK: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE Symp. Security and Privacy*, pages 511–525, 2013.

[10] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008. Proposed Standard; obsoletes RFC 3280, 4325, 4630; updated by RFC 6818 (Jan 2013). RFC 6211 explains why the signature algorithm appears twice in X.509 certificates.

[11] L. F. Cranor and S. Garfinkel, editors. *Security and Usability: Designing Secure Systems That People Can Use*. O'Reilly Media, 2005.

[12] T. Dierks and E. Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, Aug. 2008. Proposed Standard; obsoletes RFC 3268, 4346, 4366.

[13] B. Dowling, F. Günther, U. Herath, and D. Stebila. Secure logging schemes and Certificate Transparency. In *Eur. Symp. Res. in Comp. Security (ESORICS)*, 2016.

[14] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *Internet Measurements Conf. (IMC)*, pages 291–304, 2013.

[15] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. Halderman. The matter of Heartbleed. In *Internet Measurements Conf. (IMC)*, 2014.

[16] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben. Why Eve and Mallory love Android: An analysis of Android SSL (in)security. In *ACM Comp. & Comm. Security (CCS)*, pages 50–61, 2012.

[17] S. Garfinkel. *PGP—Pretty Good Privacy*. O'Reilly Media, 1995.

[18] S. Garfinkel. Using S/MIME. In *[25]*, pages 563–593, 2006.

[19] S. L. Garfinkel and R. C. Miller. Johnny 2: A user test of key continuity management with S/MIME and Outlook Express. In *ACM Symp. Usable Privacy & Security (SOUPS)*, pages 13–24, 2005.

[20] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *ACM Comp. & Comm. Security (CCS)*, pages 38–49, 2012.

[21] P. Hoffman. RFC 2634: Enhanced Security Services for S/MIME, June 1999. Proposed Standard; updated by RFC 5035 (Aug 2007).

[22] P. Hoffman and J. Schlyter. RFC 6698: The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA, Aug. 2012. Proposed Standard; updated by RFC 7218, 7671.

[23] R. Housley. RFC 5652: Cryptographic Message Syntax (CMS), Sept. 2009. Internet Standard; obsoletes RFC 3852, which itself obsoletes RFC 3369.

[24] R. Housley and T. Polk. *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructures*. John Wiley, 2001.

[25] M. Jakobsson and S. Myers, editors. *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. John Wiley, 2006.

[26] M. Just and P. C. van Oorschot. Addressing the problem of undetected signature key compromise. In *Netw. Dist. Sys. Security (NDSS)*, 1999.

[27] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communications in a Public World (2nd edition)*. Prentice Hall, 2003.

[28] S. T. Kent. Internet Privacy Enhanced Mail. *Comm. ACM*, 36(8):48–60, 1993.

[29] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Netw. Dist. Sys. Security (NDSS)*, 2015.

[30] J. Larisch, D. R. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. CRLite: A scalable system for pushing all TLS revocations to all browsers. In *IEEE Symp. Security and Privacy*, pages 539–556, 2017.

[31] B. Laurie. Certificate transparency. *Comm. ACM*, 57(10):40–46, 2014.

[32] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu. When HTTPS meets CDN: A case of authentication in delegated service. In *IEEE Symp. Security and Privacy*, pages 67–82, 2014.

[33] Y. Liu, W. Tome, L. Zhang, D. R. Choffnes, D. Levin, B. M. Maggs, A. Mislove, A. Schulman, and C. Wilson. An end-to-end measurement of certificate revocation in the web's PKI. In *Internet Measurements Conf. (IMC)*, pages 183–196, 2015.

[34] D. McCarney. A Tour of the Automatic Certificate Management Environment (ACME). *Internet Protocol Journal*, 20(2):2–14, 2017. See also RFC 8555 [4].

[35] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Free at: `http://cacr.uwaterloo.ca/hac/`.

[36] M. Naor and K. Nissim. Certificate revocation and certificate update. *IEEE J. Selected Areas in Commns*, 18(4):561–570, 2000.

[37] M. Nystrom and B. Kaliski. RFC 2986: PKCS #10—Certification Request Syntax Specification ver1.7, Nov. 2000. Informational; obsoletes RFC 2314, updated by RFC 5967.

[38] A. Oram and J. Viega, editors. *Beautiful Security*. O'Reilly Media, 2009.

[39] H. Orman. *Encrypted Email: The History and Technology of Message Privacy*. Springer Briefs in Computer Science, 2015.

[40] K. G. Paterson and T. van der Merwe. Reactive and proactive standardisation of TLS. In *Security Standardisation Research (SSR)*, pages 160–186, 2016. Springer LNCS 10074.

[41] V. Pham and T. Aura. Security analysis of leap-of-faith protocols. In *SecureComm 2011*, pages 337–355, 2011.

[42] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001.

[43] E. Rescorla. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3, Aug. 2018. IETF Proposed Standard; obsoletes RFC 5077, 5246 (TLS 1.2), 6961.

[44] S. Santesson, M. Meyers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP, June 2013. Standards Track; obsoletes RFC 2560, 6277.

[45] J. Schaad, B. Ramsdell, and S. Turner. RFC 8550: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Certificate Handling, Apr. 2019. Proposed Standard; obsoletes RFC 5750.

[46] J. Schaad, B. Ramsdell, and S. Turner. RFC 8551: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification, Apr. 2019. Proposed Standard; obsoletes RFC 5751.

[47] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL (short paper). In *Financial Crypto (FC)*, pages 250–259, 2011.

[48] E. Stark, R. Sleevi, R. Muminovic, D. O'Brien, E. Messeri, A. P. Felt, B. McMillion, and P. Tabriz. Does Certificate Transparency break the web? Measuring adoption and error rate. In *IEEE Symp. Security and Privacy*, 2019.

[49] J. Tan, L. Bauer, J. Bonneau, L. F. Cranor, J. Thomas, and B. Ur. Can unicorns help users compare crypto key fingerprints? In *ACM Conf. on Human Factors in Computing Systems (CHI)*, pages 3787–3798, 2017.

[50] S. Vaudenay. *A Classical Introduction to Cryptography: Applications for Communications Security*. Springer Science+Business Media, 2006.

[51] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J. Hubaux. The inconvenient truth about web certificates. In *Workshop on Economics of Info. Security (WEIS)*, 2011.

[52] L. Zhang, D. R. Choffnes, D. Levin, T. Dumitras, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. In *Internet Measurements Conf. (IMC)*, pages 489–502, 2014.

[53] P. Zimmermann and J. Callos. The evolution of PGP's web of trust. In *[38]*, pages 107–130, 2009.

[54] P. R. Zimmermann. *The Official PGP Users Guide*. MIT Press, 1995.

[55] M. E. Zurko. IBM Lotus Notes/Domino: Embedding security in collaborative applications. In *[11]*, pages 607–622, 2005.