Computer Security and Internet Security

Chapter 6: Certificates, Key Management and Applications

P.C. van Oorschot

May 27, 2018

Comments, corrections, and suggestions for improvements are welcome and appreciated.
Please send by email to: paulv@scs.carleton.ca

Chapter 6

Certificates, Key Management and Applications (TLS, secure email)

This chapter explains certificate management and *public key infrastructure* (PKI)—what they provide, technical mechanisms and architectures, and challenges. Two mainstream applications are considered as example uses: TLS as used in HTTPS for secure browser-server information transfer, and end-to-end encrypted email. PKI also provides core building blocks for IPsec, trusted computing, and DNSSEC (Chapter 10).

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 or access control. When passwords are used for remote authentication, they travel over a channel itself secured by authentication and confidentiality based on crypto keys. Such keys protect not only data in transit but also stored data. *Key management*—the convenient and safe distribution of such keys—includes managing not only session keys per Chapter 4, but public keys as discussed herein as well as their corresponding private keys. PKI involves data structures such as certificates and key formats, architectural components including certification authorities (CAs) and public key directories, procedures and protocols for acquiring and updating public keys and certificates, and methods for automating key management. PKI facilitates encryption and authentication, including *digital signatures* for entity authentication, data origin authentication, and to a lesser degree, non-repudiable legal signatures.

Standardization of PKI elements, e.g., X.509v3 certificate profiles, has aided interoperability and use of supporting components and software libraries across vendors and products. PKI support for public-key cryptography helps automation, security, scalability and convenience over manually-shared symmetric keys and passwords, hard-coded or manually-entered crypto keys, and symmetric-key infrastructures.

6.1 Public-key certificates and certification authorities

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 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). This is a data structure which *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 CA's signature on the certificate is its assertion that the public key belongs to the named `Subject`. Of course, this assertion is only as good as one's trust in the CA making it. Any party that relies on the certificate—any *relying party*—places their trust in the issuing CA, and requires a valid public key of that CA in order to verify the CA's signature on the certificate. Verifying the correctness of this signature is one of several steps (detailed below) that the relying party's system must carry out as part of checking the overall validity of the public key certificate.

`Version` (e.g., `X.509v3`)
`Serial-Number`
`Validity-Dates` (includes: `Not-Before`, `Not-After`)
`Subject` (includes: `Common-Name/CN`)
`Public-Key` (includes: `Public-Key-Algorithm`, `Key-Value`)
(optional) X.509 Extensions (e.g., `Subject-Alternate-Name/SAN-list`, `Basic-Constraints`, `Key-Usage`, `CRL-Distribution-Points`)
`Issuer-ID` (e.g., issuing CA)
`Signature-Algorithm` (algorithm ID)
`Digital-Signature-of-Issuer`

Table 6.1: X.509v3 certificate fields.

CERTIFICATE FIELDS. Beyond the public key, named `Subject` owning it, and CA signature, a certificate contains other fields including *attributes* that allow proper identi-

[1] Recall that Chapter 2 provides background on public-key cryptography.

fication and safe use of the public key. See Table 6.1. These include: format version, serial number, validity period, issuing-CA-identifier, and signature algorithm identifier. The `Subject` field includes `Common-Name`, intended to uniquely identify the `Subject` entity, i.e., be a *distinguished name* (DN). The public key field itself is a pair of values (`public-key-alg, key-value`); the first identifies the algorithm. The certificates most commonly used in practice, X.509v3 certificates, have both basic fields and *extension fields* (explained later). The CA signature is over all fields, i.e., the hash value digitally signed encompasses all bits of all certificate fields, for integrity protection.

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 6.5).

ACQUIRING A CERTIFICATE. Protocols have been standardized for an end-entity to request a certificate from a CA.[2] 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.

[2] See RFC 4210 [1], which allows certificate requests per the earlier PKCS #10 of RFC 2986 [28].

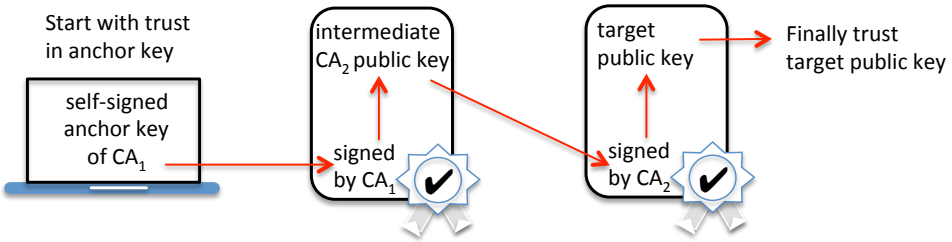

Figure 6.1: Certificate chain validation.

6.2 Certificate chain validation and certificate extensions

CHAINS AND INTERMEDIATE CAS. CAs generate certificates for end-entities, e.g., for human users in the case of email certificates, web servers in the case of TLS certificates. CAs also create certificates for the public keys of other CAs. The other CA is sometimes called an *intermediate CA*, e.g., when the first CA is a trust anchor or atop a hierarchy (as explained later). Figure 6.1 shows the concept of a *certificate chain*.

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`];
2. has not been revoked (Section 6.3);
3. has a signature that verifies (mathematically), using the signing CA's public key;
4. is signed by a CA who's public key is (available and is) itself trusted;[3]
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.[4]
6. use is consistent with all constraints specified in certificate extension fields or policies (e.g., path length, name constraints, key usage restrictions—see below).
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. See Figure 6.1.

[3] See discussion of *trust anchors* immediately below.

[4] This is an example of the principle of Response-Integrity (Chapter 1).

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. Such hash values may be represented as hexadecimal strings, or images uniquely derived from them. The out-of-band channel might be paper over 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 signalling* means sending control data over the same channel as voice data.

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 data. 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 determines which TLS (server) certificates the browser, and thus 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 6.2). The browser software may reject it, or present a dialogue allowing the user to accept it as "trusted" despite the software not recognizing it as such—in this case ideally also presenting the user information allowing a certificate fingerprint check (as above; Figure 6.9 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 principle—users should not be asked to take decisions that they do not have sufficient information to make properly—but

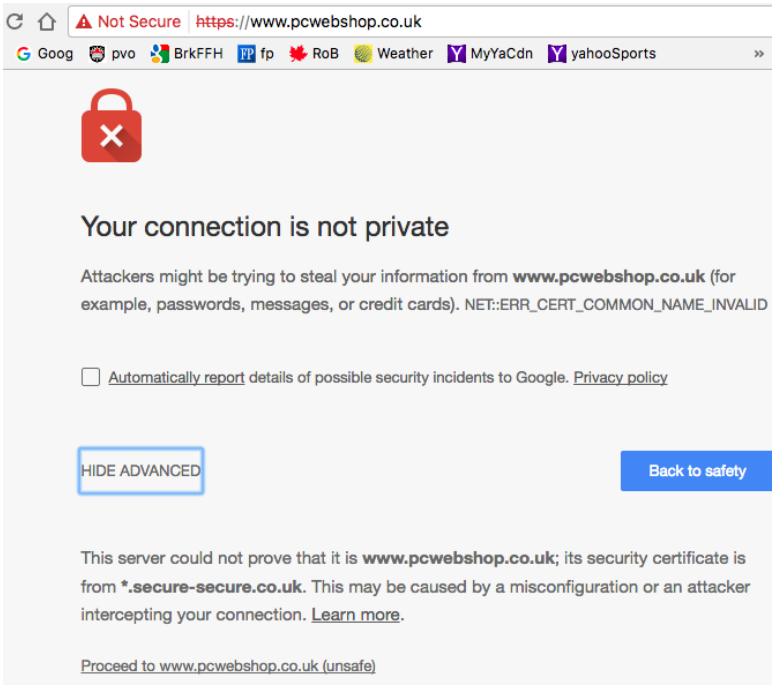

Figure 6.2: Self-signed site certificate—browser warning (Chrome 56.0.2924.87). The "Back to safety" tab discourages clicking-through to the site despite the browser being unable to verify the certificate chain (this happens when one or more certificates are signed by CAs unrecognized by the browser, i.e., not verifiable using the browser's trust anchors).

it is a common short-cut 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 electronically from a remote party, without any cross-check or assurance that it is authentic, this is called *trust on first use* (TOFU), or *blind* TOFU to emphasize the risk and lack of cross-check. Some software interfaces ask the user if 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, and checked for a match 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 call it "check on first use".

A common use of TOFU is in SSH (Chapter 10), e.g., configured for user-password, server public key. On first visit to an SSH server, an SSH client receives the server public key and is given an option to accept it (after optionally cross-checking its fingerprint). If accepted, the user enters a password for authentication to the server, the password transmitted over a secure channel set up using the server's public key.

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 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 it 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 (the key cannot be used to verify chain links).

- `Key-Usage`: allowed uses of key, e.g., for signatures, encryption, key agreement, CRL signatures (below). 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`: may include for example an email address, domain name, IP address, URI, or other name forms. If the `Subject` field is empty, the alternate name must be present and marked as a critical extension.

- `Name-Constraints`: in CA-certificates (below), allow CA control of `Subject` names in subsequent certificates when using hierarchical *namespaces*. By specifying prefixes in name subtrees, specified namespaces can be excluded, or permitted.

CROSS-CERTIFICATE PAIRS. ITU-T X.509 standardized a data structure for a pair of *cross-certificates* between CAs, i.e., each signing a certificate for the public key of the other—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 to enable secure email between their communities; the structure can aid discovery and assembly of certificate chains. Single (uni-lateral) 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 additional 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?)

6.3 ‡Certificate revocation

Certificates have a pre-defined 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 pre-defined 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 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

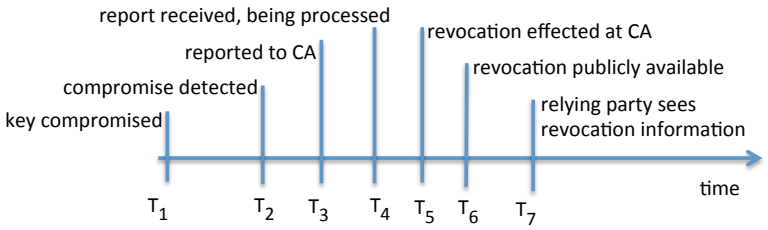

Figure 6.3: Revocation timeline: from compromise to visibility. Possible delays at T4 are mechanism-dependent, e.g., CRLs are typically issued at periodic intervals. A benefit of OCSP mechanisms (over CRLs) is to remove the T6-to-T7 delay.

instantaneously broadcast revocation information may reduce legal liability, but overlooks other aspects. Consider the event timeline of a private key compromise in Figure 6.3.

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 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 train of thought leads to considering not only entirely eliminating revocation, but possibly even signed certificates, instead relying on a trusted key server to serve only valid keys. This approach is best suited for 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 clearing-house 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 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, an attacker blocking revocation services may cause a revoked certificate to be relied on.[5]

6.4 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.

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 human users are trustworthy.

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 who the system is designed to give control to.

TOOLBOX SHAPED TO MEET GOALS. Standards and data structures associated with ITU-T X.500 and X.509 certificates, and customizations via IETF RFCs, provide a toolbox allowing different communities/applications to build customized trust models suiting specific needs and ecosystems. Different trust models should be expected for automative 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.

[5] Note that this violates the principle of Safe-Defaults (Chapter 1): systems should fail closed (not open).

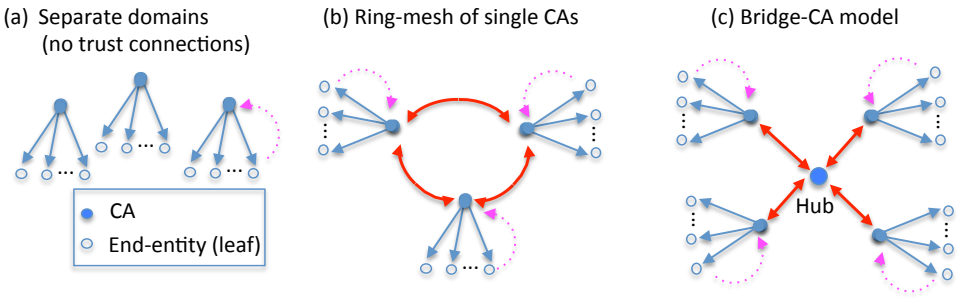

Figure 6.4: Model I: Single-CA systems and linking them. Solid arrow points from certificate signer to certificate subject. Double arrow indicates CA-certificates in both directions (i.e., cross-certificate pair). Dotted arrow indicates trust anchor. Case (b) shows that for $n = 3$ CAs, a ring of cross-certificate 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 neighbours 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.

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 current end-to-end secure *instant messaging* systems (e.g., WhatsApp Messenger).

Figure 6.4 illustrates simple topologies for linking single-CA domains. 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 6.4(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 6.4(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.

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. This model was 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 from order n^2 to n. Figure 6.4(c) shows this with single-CA sub-systems; Model III's multi-CA sub-systems can likewise

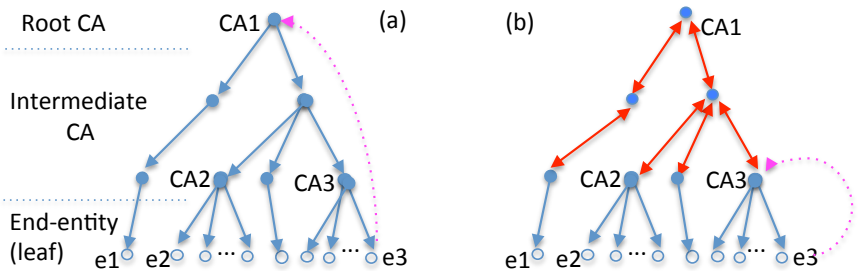

Figure 6.5: Model II, CA trust models: (a) Strict hierarchy. (b) Hierarchy with reverse certificates. Nodes are certificates. Solid arrow points from certificate signer to subject. Double-arrow means CA-certificates in both directions. Dotted arrow shows trust anchor.

be bridged. Note the bridge CA is not a trust anchor for any end-entity (thus not a root).

MODEL II: STRICT HIERARCHY & HIERARCHY WITH REVERSE CERTIFICATES. A *strict 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 Figure 6.5(a). CAs at a given tree level create 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 are 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 Figure 6.5(a), trace out the trust chain path from e3 to e2; then do so in Figure 6.5(b).

A relaxation of this model 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 6.5(b) shows this 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 Figure 6.4, 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. For convenience, 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

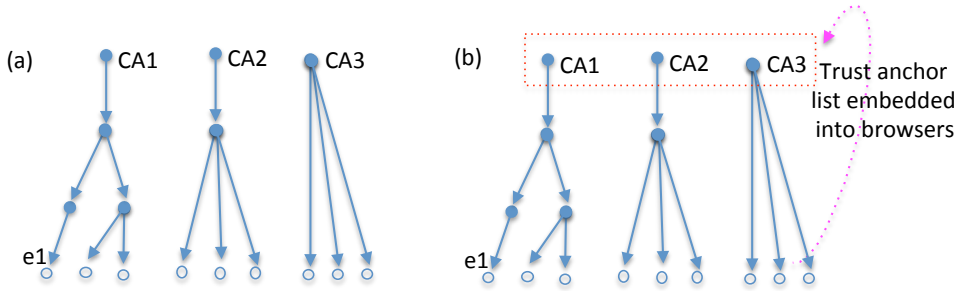

Figure 6.6: Model IV: Browser trust model. Built from multiple disjoint hierarchical trees (a), there are no CA cross-certificates in the browser trust model (b) and leaf certificates are for servers (e.g., e1). Solid arrow points from certificate signer to subject.

2) pairs of roots cross-certifying, is possible but 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.

MODEL IV: FOREST OF HIERARCHICAL TREES. 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*. See Figure 6.6. End-entities get as trust anchors not just a single root key, but a (typically large) set of public keys corresponding to a collection 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*, used to authenticate not end-users, but rather to allow user 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 (Figure 6.7), complete networks, hierarchies, ring-meshes and bridge-CAs. While not recommended, arbitrarily complex trust graphs are allowed—with complexity 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

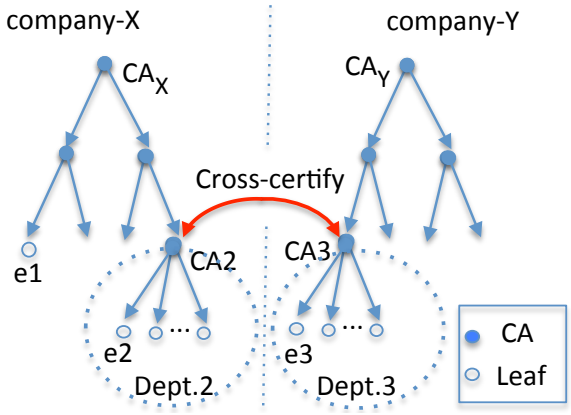

Figure 6.7: Model V: Decentralized CA trust. Cross-certifying peer departments lower in a hierarchy allows finer-grained trust peering than cross-certifying at the root. Solid arrow points from certificate signer to certificate subject. See text for further explanation.

bottom-up, rather than fully-planned before any roll-out begins. Commercial products have focused on tools suitable for use within and between corporations and government departments, i.e., *enterprise products*. Companies may build a PKI first within a small department, then a larger division, then perhaps across international arms, and perhaps later wish to be able 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 6.7). 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 more fine-grained alternative, suppose the motivation stems from e2 being in a division Dept2, which 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 (see Figure 6.7). 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 6.2) 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 which communicate regularly results in shorter (simpler) chains.

MODEL VI: USER-CONTROL CERTIFICATE TRUST MODEL. 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 of end-entities. This is the PGP model, proposed circa 1995 for secure email among small groups of technically-oriented users. Further discussion is deferred to Section 6.6.

CROSS-CERTIFICATES VS. TRUST ANCHOR LISTS. As highlighted in this section, two main aspects distinguish PKI trust architectures, defining how trust flows between trust domains (*communities of trust*), and thus between end-users:

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).

6.5 TLS web site certificates and CA/browser trust model

The preceding has discussed generic aspects of certificates and PKI. We now discuss their specific application to TLS—building on the browser trust model of Figure 6.6.

TRANSPORT LAYER SECURITY. *TLS* is the world's most widely deployed security protocol and *de facto* means for securing web browser-server traffic. While use of TLS/SSL by HTTP yielding HTTPS was the primary motivation, its 1994 predecessor *SSL* was designed with the intent that almost any protocol using TCP could be easily modified to be run "over" TLS/SSL to add security. By 2000 it had already been used to add security to FTP (file transfer), SMTP (email), telnet (remote terminals) and LDAP (directory access). The two original security goals were encryption of traffic between end-points (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 6.2) and concludes that from its viewpoint, the certificate is valid. A certificate trusted by one relying party might not be by another.

GRADES OF TLS CERTIFICATES. As discussed (Section 6.1), 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 by minimal 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, e.g., Common Name, Organizational Unit, Organization, and Country. Most end-users notice no difference between DV and OV certificates (differences in browser cues, if any, are too small or not understood).

EV CERTIFICATES. CA checks carried out before issuing *Extended Validation* (EV) certificates include the following. Verification of real-world existence of the legal entity named as `Subject`; registration of this organization in government-recognized databases; confirmation of a physical, operational existence matching the location indicated in the certificate; confirmation of the identity of the requesting individual, and their authority to act on behalf of the organization; verification of 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.

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.[6] EV guidelines require that CAs publish operational policies; conformance is third-party audited. On browser user inter-

[6] For official documents, see [4, 5].

faces, EV certificates result in the URL bar/lock icon colouring differently (this varies by browser), and a URL bar showing the `Subject` name and jurisdiction (e.g., preceding the DN); see Figure 6.8. EV certificates were motivated by loss of confidence as low-assurance DV certificates emerged—for example, browser-trusted DV certificates on phishing sites result in the same visual assurances to end-users as non-phishing sites, including the visual TLS closed-lock icon in browser URL bars. Absent additional means by browsers to effectively communicate differences between EV and (OV, DV) certificates, what added value EV certificates deliver to end-users remains unclear.

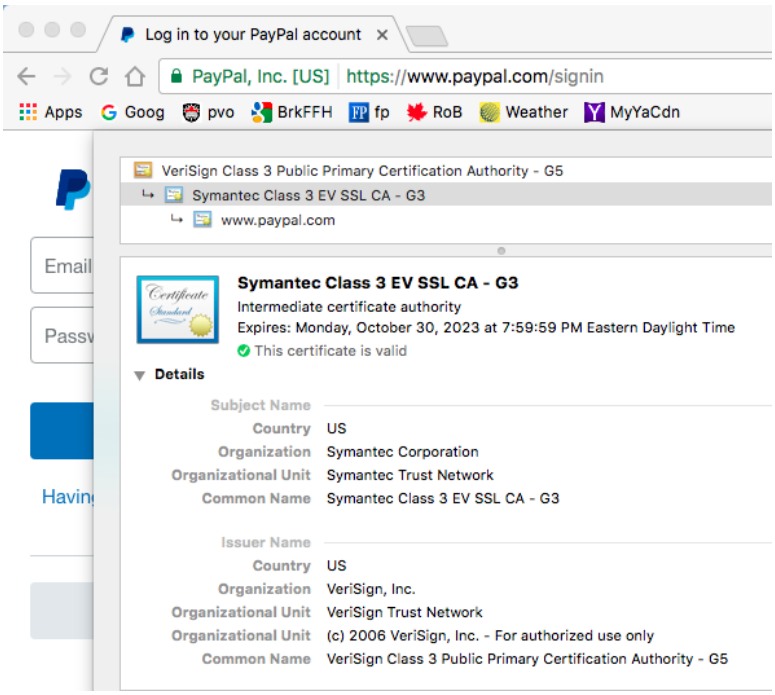

Figure 6.8: Intermediate CA EV certificate (Chrome 56.0.2924.87 browser). The URL bar displays organization details of the server site "Paypal, Inc. [US]" between the lock icon and "https:". The grey-highlighted "Symantec Class 3 EV SSL CA - G3" shows this as the middle of 3 certificates in a chain; 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].

SELF-SIGNED TLS SERVER CERTIFICATES. Self-signed TLS certificates were once common, before free DV certificates were easily obtained; non-commercial sites often preferred to avoid third-party CAs and related costs. Browser dialogues are now worded to discourage or entirely disallow this (Figure 6.2). TOFU should be strongly discouraged for non-leaf certificates, due to trust implications (private keys corresponding to CA certificates can sign new certificates). TOFU is one method for distributing email

leaf certificates—incoming email offering a sender's encryption and signature public keys.

Example *(Number of TLS CAs).* A March 2013 study observed 1832 browser-trusted CA signing certificates, including both trust-anchor CA and intermediate-CA certificates, associated with 683 organizations across 57 countries [11].

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 errors). Discuss practical challenges related to *domain mismatch errors*, i.e., checking that the Common Name subfield of a certificate `Subject` matches the domain a browser is visiting using TLS. (Hint: see [37].)

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

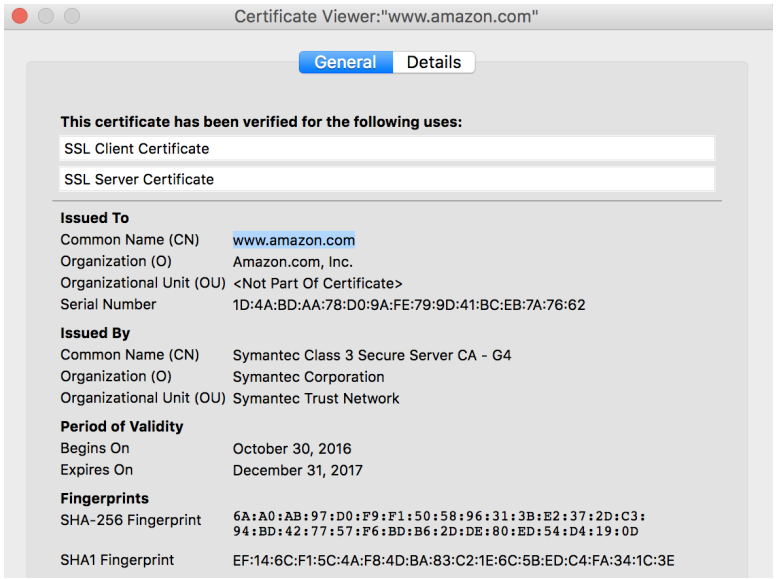

Figure 6.9: TLS site certificate example: general tab (from Firefox 55.0.1 UI). The UI tool, under **Issued To**, displays the certificate `Subject` with subfields CN (giving domain name `www.amazon.com`), O and OU; compare to Figure 6.10. **Issued By** indicates the certificate-signing CA. Under **Fingerprints** are the hexadecimal values of the certificate hashed using algorithms SHA-256 and SHA1, to facilitate a manual security cross-check.

BROWSER INTERACE DIALOGUES ON CERTIFICATES. Information related to TLS certificates can be found, e.g., by clicking on the closed-lock icon in a browser's URL bar. Figure 6.2 shows a warning dialogue upon encountering an (untrusted) self-signed site certificate. Figure 6.9 gives high-level information about a site certificate. Figure 6.10 gives detailed information; the interface allows exploration of the certificate chain.

BROWSER TRUST MODEL ISSUES. It is generally recognized that the browser trust model as currently used, has significant limitations:

1. *Rogue certificates* are those created fraudulently (e.g., by an untrustworthy CA or using the private key of a compromised CA), or otherwise unauthorized by their subjects. They can be used in *certificate substitution* attacks, enabled by the browser trust model allowing any CA to issue a certificate for any domain, treating all trust anchors as equal. Fraudulent site certificates will be recognized as valid by all browsers housing a corresponding CA public key as a trust anchor. An intermediate CA issued a certificate by a rogue CA can also create fraudulent certificates. The browser trust model is thus *fragile*, in that a single rogue CA can undermine the entire system, regardless of the strength of other CAs.

2. *TLS-stripping* attacks are hard to stop. In these active attacks, a legitimate browser, directed to switch from HTTP to HTTPS, is interfered with such that the upgrade request is not executed. Unsecured HTTP leaves transferred data vulnerable.

3. Revocation remains poorly supported by browsers. They commonly "fail open" when revocation services are unavailable (i.e., revocation is essentially ignored).

4. *Trust agility*—the ability for end-users to alter trust anchors—is poorly supported. Browser and operating system vendors commonly populate hundreds of trust anchors, while most users actively rely on very few.

INVISIBLE INTERMEDIATE CAS. Detection of rogue certificates is complicated by intermediate CAs being largely invisible to users. The resulting lack of accountability enables related *compelled certificate* attacks, whereby a trust anchor CA is coerced to issue intermediate CA certificates by shady organizations governing or regulating them, to facilitate surveillance and middle-person attacks.

As a case study of flaws in a system in use for 25 years, the above issues remain interesting even should they be resolved in the future.

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 names and the *DANE* protocol: DNS-based Authentication of Named Entities. Describe how DANE works, and its relationship to DNSSEC. (Hint: see RFC 6698 [15].)

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: see [12].)

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-sharing, 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 *cruise-liner certificates*. (Hint: see [24, 7].)

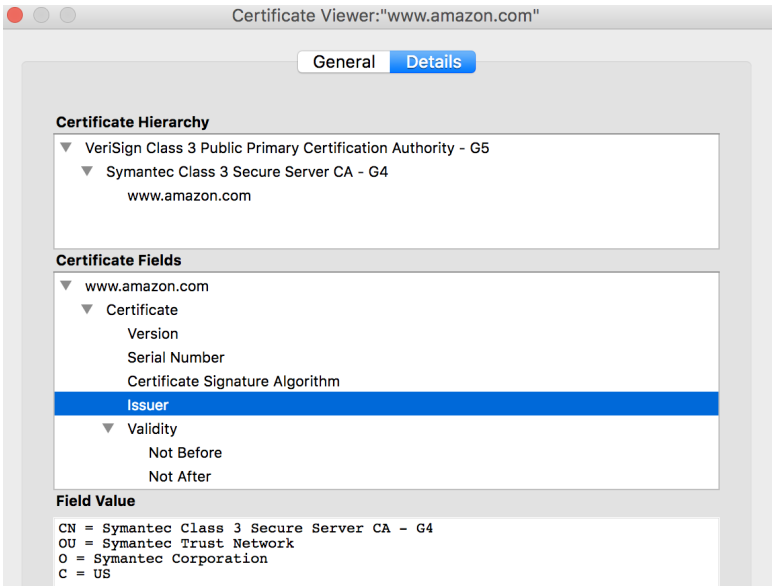

Figure 6.10: TLS site certificate example: details tab (from Firefox 55.0.1 UI). The **Certificate Hierarchy** segment displays the certificate chain. As the user scrolls through the middle portion to access additional fields, a selected field ("Issuer" here) is highlighted by the display tool and its value is displayed in the lower portion. Notations OU (organizational unit), O (organization), and C (country) are remnants of X.500 naming conventions.

6.6 Secure email: S/MIME, PEM, PGP

In *end-to-end secure email*, messages are encrypted and digitally signed on originator devices, and decrypted on recipient devices. This provides an application study for PKI and certificate management that is complementary to TLS. We discuss three initiatives.

1. S/MIME (Secure MIME). When used with centralized certificate management, it is suitable for enterprise users including government departments and corporations.

2. PEM (Privacy Enhanced Mail). With a one-root PKI hierarchy, it is of historic and technical interest as the first comprehensive secure email effort (circa 1990-1995).

3. PGP (Pretty Good Privacy). With ad hoc trust management for highly-technical users, its 1991 design suits private individuals and relatively small, static groups.

EMAIL TRANSPORT. A *mail user agent* (MUA) transfers mail to the originating user's sending mail server. 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 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 retrieved). These protocols allow mail storage on servers, clients, or both—thus end-to-end secure email designs must consider storing delivered email in plaintext, or encrypted (and if so, whether to re-encrypt by means independent of mail transport).

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 body section—inserting into the body, for example, a new 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 would then display the interior security header and encrypted body as labelled fields followed by meaningless ASCII characters. Commonly, the originating client uses a symmetric key k (*message key*) to encrypt the plaintext body. The plaintext body is also hashed and digitally signed. The security header includes fields providing:

- 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 copy, and the private key needed);
- an identifier for the symmetric encryption algorithm used, plus any parameters;
- the originator's digital signature, plus an identifier of the signing algorithm;
- an identifier of the originator's public key to verify the signature (the public key itself could be provided, in case the recipient does not have a copy of that key);
- any further information needed for signature verification, optionally including a chain of certificates to allow trust in the signing public key.

It is common to include a copy of k encrypted under the originator'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. Various possibilities for the PKI model and trust anchors can facilitate trust in public keys of both the intended recipient, and originator.

Exercise (Email transport). Sketch a block diagram for regular email transport.

Exercise (Security header). Sketch a field layout for the security header above.

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

Exercise (Order of signing and encrypting). It is common for commercial products to first compute a digital signature, and then encrypt both the signature and mail body. What advantage does this offer, over first encrypting the mail 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 supporting HTML messages 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. *Key escrow* architectures can provide plaintext access to gateway servers, e.g., by retrieving an escrowed copy of the mail originator's decryption private key, at the cost of performance, defeating end-to-end encryption, and risks due to added complexity and attack surface.

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 CA-based 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*, 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 needs, and with access to suitable certificate directories. Enterprise PKI trust models (Section 6.4) 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 one-size-fits-all solutions are unlikely. One option is: regress to 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: DISTINGUISHING FEATURES.** 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. S/MIME superseded PEM.

‡**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 in 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 it by phone, can 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 one centralized CA by 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 CHALLENGES.** PGP's core architectural design, reflecting its original objectives, should not be expected to match general secure email requirements. Specific challenges include: 1) The manual, ad hoc nature of the web of trust does not scale to larger communities; 2) User management of trust requires technical expertise 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; and 3) The non-centralized model leaves revocation of PGP keys unresolved—end-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.

6.7 End Notes

For authoritative treatments of X.509 certificate-related security, PKI architecture and trust models, see Housley [17] (including the U.S. Federal Bridge CA project) and Adams [2]; see also Kaufman[19] and Menezes [26]. RFC 5280 [9] 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 [34]; 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 [27, 2]. The revocation timeline of Figure 6.3 is based on Just [18].

For TLS/SSL history, see Rescorla [33]. RFC 5246 [10] specifies TLS 1.2; TLS 1.3 (draft 28, 21 Mar 2018) has major changes. Clark [8] summarizes challenges with HTTPS and its certificate trust model; Liu [25] measures web PKI support for revoking TLS certificates. Larisch [22] proposes a method for pushing such revocations to browsers. The TLS Heartbleed incident revealed an inability to handle massive-scale TLS certificate revocations [38, 12]. Saghoian [35] discusses *compelled certificates*. Liang [24] and Cangialosi [7] discuss how *content distribution networks* (CDNs) and site host providers interact with TLS. Laurie compares *Certificate Transparency* [23] to alternatives for improving the CA/Browser trust model. On addressing TLS stripping (by HTTPS strict transport security/*HSTS*) and rogue certificates (by *public-key pinning*), see Kranch [21].

Orman [30] summarizes technical challenges in implementing S/MIME and PGP; see also Kaufman [19]. S/MIME uses the Cryptographic Message Syntax (CMS) of RFC 5652 [16]; for S/MIMEv3.2 see RFCs 5750 and 5751 [31, 32] (cf. RFC 2634 [14]). For PEM, see Kent [20]. Zimmermann [40] is the definitive PGP reference; see also Garfinkel [13], RFC 4880 [6] for OpenPGP message formats (also `https://www.openpgp.org/`), comments from Vaudenay [36, §12.4], and more recent views of proponents [39].

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. *Commun. ACM*, 57(10):47–55, 2014.

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

[5] 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`.

[6] 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.

[7] 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 Conf. Computer and Comm. Security (CCS)*, pages 628–640, 2016.

[8] 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.

[9] 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).

[10] 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.

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

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

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

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

[15] 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.

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

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

[18] M. Just and P. C. van Oorschot. Addressing the problem of undetected signature key compromise. In *ISOC Symp. Network and Distributed System Security (NDSS)*, 1999.

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

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

[21] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air:An empirical study of strict transport security and key pinning. In *ISOC Symp. Network and Distributed System Security (NDSS)*, 2015.

[22] 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.

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

[24] 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.

[25] 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 Conference (IMC)*, pages 183–196, 2015.

[26] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Freely available online.

[27] M. Naor and K. Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000.

[28] 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.

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

[30] H. Orman. *Encrypted email: The history and technology of message privacy*. Springer Briefs in Computer Science, 2015.

[31] B. Ramsdell and S. Turner. RFC 5750: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling, Jan. 2010. Proposed Standard; obsoletes RFC 3850.

[32] B. Ramsdell and S. Turner. RFC 5751: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification, Jan. 2010. Proposed Standard; obsoletes RFC 3851.

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

[34] 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.

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

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

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

[38] 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 Conference (IMC)*, pages 489–502, 2014.

[39] P. Zimmermann and J. Callos. Chapter 7: The Evolution of PGP's Web of Trust. In *[29]*, pages 107–130, 2009.

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