Lecture Notes for CS-578 Cryptography (SS2006) Prof. Michael Backes

11. Certificates

Lecture 16 Saarland University

We already saw in Chapter 5 that the secure exchange of symmetric keys is a difficult task, since both privacy and authenticity of the keys have to be ensured. In particular, this served as our major motivation for coming up with public-key cryptography, where secure exchange of public keys only requires authentic transmission of the keys. This chapter is finally dedicated to showing how such an authentic transmission can be achieved.

The main idea can be roughly summarized as follows: One considers a central authority, a so-called *certification authority (CA)*, that is trusted by every user. We assume that this authority correctly checks the identity of every person that is willing to create a key pair, and then issues a so-called certificate. This certificate contains a signature over the public key as well as the identity of the key's owner, thus binding the public key to a specific real-life person or organization. This signature is created using the secret signing key of the CA. Ideally, every user knows the corresponding public key of the CA so that the validity of every certificate can be locally tested by everybody. The resulting infrastructure is often called a *public-key infrastructure* (PKI).

11.1 (Key-) Certificates

A certificate binds a public key to a real-life person or an organization. Intuitively, certificates constitute signed statements of the form

> "I, Max Mueller, take responsibility for the secret key which belongs to the public key 100101100101010...0101. This is affirmed by the certification authority CA."

Technically speaking, a certificate consists of a digital signature over the user's name and additional unique identification information, the issuer identification, a serial number, an expiration date, the user's public key, as well as additional technical information and various optional extensions. We show two examples of certificates in Figure 11.1; both examples are taken from `www.wikipedia.org`.

The certification authority itself owns a certificate which is pre-distributed to all participants, and which is considered trusted. Certificates of important CAs are usually included immediately in web browsers and in other relevant software. Note that the CA itself is an offline entity in the sense that it only takes action when issuing a certificate, but it does not participate in the actual distribution and verification of the certificate in the future. Note further that this constitutes an important advantage over key distribution centers for symmetric keys, as these centers have to be online all the time.

The general idea how certificates are used can be summarizes as follows: A user Alice first creates a key pair, presents her public key to the CA, and proves her identity (either using her passport, a notarial document, etc.). If the CA can successfully verify Alice's identity, the CA issues a certificate on her public key and identity, and hands this certificate over to Alice. If Bob wants to send a message to Alice that is encrypted with this public key, Bob and Alice can achieve this as follows:

1. Alice first sends her certificate to Bob. (As it is Bob that would like to send a message to Alice, this step is typically triggered by Bob in practice, i.e., Bob would request the certificate first.)
2. Upon receiving the certificate, Bob verifies the validity of the certificate's signature with respect to the public key of the CA (recall that we assumed so far that the public key of the CA is known to everybody) and checks if the signature is over the correct data, i.e., that the identifying information obtained from the certificate indeed fit to Alice.
3. If this verification succeeds, Bob retrieves the public key of Alice from the certificate, encrypts the desired message using this key, and sends the ciphertext to Alice.

11.1.1 Single Domain Certification Authority

So far we assumed that there exist a single CA whose public key is known to everybody. This assumption turns out to be justified in practice if certification is only performed within a single enterprise; we sketch this scenario in Figure 11.2. While the existence of a single CA is indeed the easiest conceivable case, an authority that is globally trusted by everybody and that verifies the identity of every human seems hardly achievable if we consider global certification and communication over the Internet. These and similar practical concerns can be tackled by the following constructions.

11.1.2 Hierarchical CAs

A possible solution is to structure CAs hierarchically as sketched in Figure 11.3. There is still one central authority, here called *Root-CA*. However, this Root-CA does not certify users but it certifies other certification authorities. These secondary CAs may then either certify users or again other CAs, thereby further expanding the hierarchy. In such an hierarchical scenario, each user needs to own a valid certificate of the public key of the Root-CA. All other certificates are sent along with the user certificates when they are needed, thus enabling the deployment of a large number of secondary CAs without decreasing performance.

Suppose Bob wants to verify the certificate of Alice in the example presented in Figure 11.3. Along with her certificate $cert_{\mathsf{Alice}}$, she sends the certificate $cert_1$ to Bob. Bob then first verifies the validity of the certificate $cert_1$ using the public key of the Root-CA (i.e., using the root certificate he owns). If this verification succeeds, he verifies the validity of the certificate $cert_{\mathsf{Alice}}$ with the key obtained from $cert_1$. If the hierarchy spans more then two layers, this step is applied iteratively.

11.1.3 Cross-Certification

One does not necessarily need a single root-CA for linking two CAs. Instead one can allow several CAs to cross-certify each other. For instance, two CAs cross-certifying each other yield certificates $cert_{12}$ and $cert_{21}$, cf. Figure 11.4.

If Bob wants to verify the validity of the certificate of Alice, Alice first has to determine Bob's CA. Then she sends her certificate $cert_{\mathsf{Alice}}$ along with the right cross-certificate $cert_{12}$ to Bob. Bob then verifies the validity of $cert_{12}$ using the public key from CA_2's root certificate, which he knows, and then verifies the validity of Alice's certificate with the key obtained from $cert_{12}$.

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town,
               O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft,
                CN=www.freesoft.org/Email=baccala@freesoft.org
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```

```
Certificate:
   Data:
       Version: 3 (0x2)
       Serial Number: 1 (0x1)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Aug  1 00:00:00 1996 GMT
           Not After : Dec 31 23:59:59 2020 GMT
       Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
                OU=Certification Services Division,
                CN=Thawte Server CA/Email=server-certs@thawte.com
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c:
                   68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da:
                   85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06:
                   6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2:
                   6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b:
                   29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90:
                   6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f:
                   5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36:
                   3a:c2:b5:66:22:12:d6:87:0d
               Exponent: 65537 (0x10001)
       X509v3 extensions:
           X509v3 Basic Constraints: critical
               CA:TRUE
   Signature Algorithm: md5WithRSAEncryption
       07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9:
       a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48:
       3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88:
       4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9:
       8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5:
       e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9:
       b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e:
       70:47
```

Figure 11.1: A Certificate and the Corresponding Root Certificate in Informal Notation

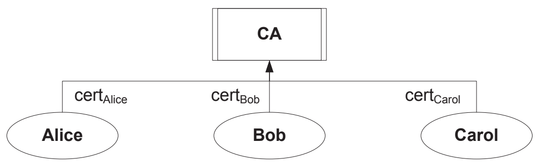

Figure 11.2: Single Domain CA

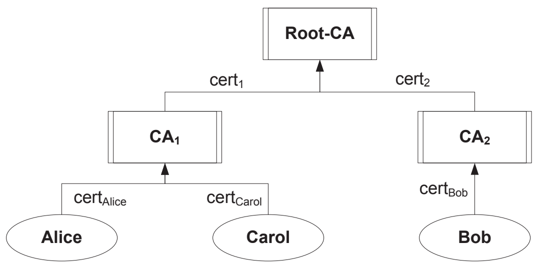

Figure 11.3: Hierarchical CAs

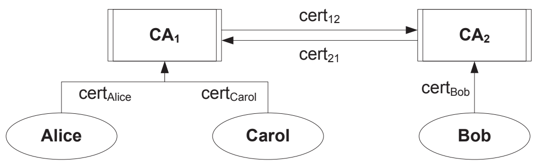

Figure 11.4: Cross-certification

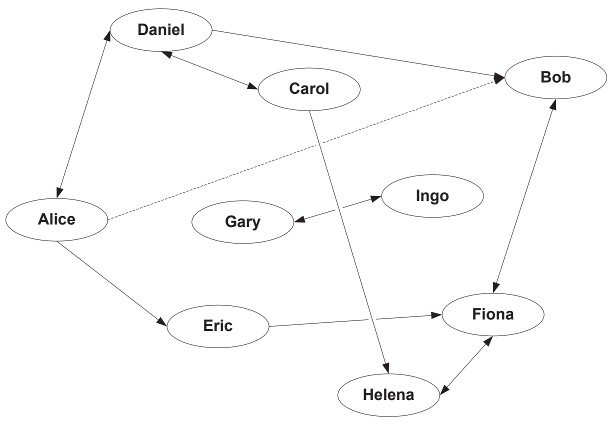

Figure 11.5: An Example Web of Trust

11.1.4 Web of Trust (PGP)

This idea can be further developed to yield a so-called *web of trust* as follows: Each user typically trusts several other users, say friends or colleagues. These in turn trust other users, and so on. Now one can hope (and in practice this seems to work pretty well) that if Bob can identify one or even multiple paths between Alice and himself, the public key of Alice can be trusted. This is illustrated in Figure 11.5, where a (solid) arrow from a node A to a node B means that B certified the public key of A. Suppose Bob wants to verify Alice's certificate. There exist two paths from Alice to Bob:

$$\text{Alice} \to \text{Daniel} \to \text{Bob},$$

and

$$\text{Alice} \to \text{Eric} \to \text{Fiona} \to \text{Bob}.$$

Consider the first path. Bob trusts Daniel who himself certified Alice's key. Then Bob verifies the certificate $cert_{\mathsf{Daniel} \to \mathsf{Alice}}$ with Daniel's public key. This gives him Alice's public key. To counter the case that one party in such a path is corrupted, one requires to have at least two (or better even multiple) different paths to establish trust in a user's public key.

11.1.5 Maurer's Scheme

Maurer's scheme provides a general model for determining "whom we trust". For every user B consider the following four predicates:

- $\mathsf{Aut}(X)$: B believes that the public key of X is authentic,
- $\mathsf{Cert}(X, Y)$: B has a certificate issued by X for the public key of Y,
- $\mathsf{Trust}(X, 1)$ (trust of level 1): B trusts X to correctly verify every user's identity before issuing certificates for them,
- $\mathsf{Rec}(X, Y, 1)$ (recommendation of level 1): B has an (attribute-) certificate where X says that he/she trusts Y to correctly verify every user's identity before issuing certificates for them.

Inductively, one defines predicates for higher levels of trust:

- $\mathsf{Trust}(X, i)$ (trust of level i): B trusts X to correctly verify recommendations of level $i-1$,
- $\mathsf{Rec}(X, Y, i)$ (recommendation of level i): the corresponding (attribute-) certificate.

Initially a user knows public keys for some users X ($\mathsf{Aut}(X) = \mathsf{true}$), he knows some certificates ($\mathsf{Cert}(X, Y) = \mathsf{true}$), he trusts some users X for some level i ($\mathsf{Trust}(X, i) = \mathsf{true}$) and got some recommendations for some level i ($\mathsf{Rec}(X, Y, i) = \mathsf{true}$). With the following rules he can deduce further trust:

- $\mathsf{Aut}(X) \wedge \mathsf{Trust}(X, 1) \wedge \mathsf{Cert}(X, Y) \rightarrow \mathsf{Aut}(Y)$
- $\mathsf{Aut}(X) \wedge \mathsf{Trust}(X, i+1) \wedge \mathsf{Rec}(X, Y, i) \rightarrow \mathsf{Trust}(Y, i)$.

One says that B trusts the keys of user Q if he can deduce $\mathsf{Aut}(Q)$ with the above rules from the initially given set of predicates. (However, this presupposes that trust is indeed transitive which is questionable.)

11.1.6 Certificate Revocation

Eventually it may happen that a secret key gets compromised. We then need a mechanism to invalidate the corresponding certificates; one says that these certificates are *revoked*. Such a revocation can be seen as a statement, signed by the CA, that certain explicitly specified certificates should be considered invalid. This statement needs to be signed, as otherwise an attacker could easily revoke any certificate, resulting in a denial-of-service attack. A crucial point is how these revocation statements reach the users, which should be prevented from accepting revoked certificates.

Certificate Revocation Lists (CRL) The conceptually simplest method is to publish a list of all revoked certificates, a so-called *certificate revocation list* (CRL). This list is signed by the CA and contains serial numbers of all certificates that are revoked. The main problem with this approach is that these lists can get very long. In practice, this is circumvented by using so-called *incremental CRLs*, where only new revocations are added.

Online Certificate Status Protocol (OCSP) A suitable way to circumvent the problem of having long CRLs is to offer an online verification service for certificates, i.e., an *online verification authority* (VA) is provided that checks, upon a user's request, if a given certificate was revoked or not. Provided that the number of VAs is small, they can be synchronized in little time. The main problem with this solution is the high load of the VAs, and as all these VAs have to be completely trustworthy, replicating VAs in a significantly large number to even out the load is hardly possible.

Certificate Revocation Trees (CRT) A nice remedy to this problem is the following one: We modify the scheme such that there is only one Root-VA, which everybody has to trust, and a large number of VAs that do not have to be trusted. Consequently, one may have a large number of VAs, which results in a scalable solution. However, we somehow have to guarantee that a malicious VA cannot trick an honest user into accepting revoked certificates.

This can be achieved as follows. Remember that certificates have a serial number, which is used to address the certificate. Let $C_1, \dots, C_4$ be the serial numbers of the revoked certificates, ordered such that $C_i < C_{i+1}$. The Root-CA builds a tree as depicted in Figure 11.6, i.e., it hashes C_i obtaining H_i, and then constructs a hash tree yielding a top node H_7. Finally, the Root-VA signs the value H_7 of the root node and the depth of the tree yielding a signature $\mathsf{S}(sk_{\mathsf{Root\text{-}VA}}, (\text{tree-depth}, H_7))$. Then the following data is sent to the VAs, including the potentially malicious ones:

$$CRT = (C_1, C_2, C_3, C_4, \mathsf{S}(sk_{\mathsf{Root\text{-}VA}}, (\text{tree-depth}, H_7))).$$

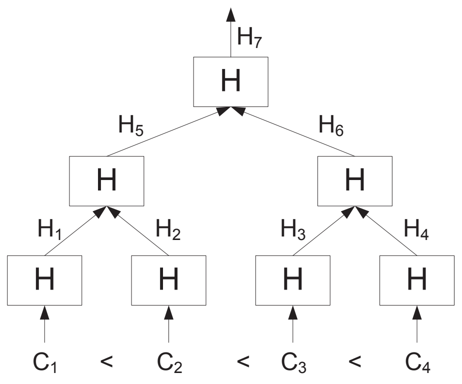

Figure 11.6: CRT

Now assume that a user queries a VA for a certificate with serial number C. Let us first consider the case that the certificate was revoked, for example $C = C_2$. Then the VA proves this as follows. Let us first note that given $C_1, \dots, C_4$, the VA can recompute the entire tree of hash values. The VA sends $(\mathsf{S}(sk_{\textsf{Root-VA}}, (\text{tree-depth}, H_7)), H_1, H_6)$ to the user, which is enough information for the user to recompute the root node, which he can then compare to the root node contained in the signature of the Root-VA.

If a user asks for a certificate with serial number C which was not revoked, then the VA proves this as follows. It identifies the largest serial number which is smaller than C, and the smallest serial number which is larger than C, e.g., $C_1 < C < C_2$. It sends $(\mathsf{S}(sk_{\textsf{Root-VA}}, (\text{tree-depth}, H_7)), C_1, C_2, H_6)$ to the user. Again the user can recompute the root node of the hash tree and test if it equals the value H_7 in the signature of the Root-VA.

However, if the VA tries to cheat, e.g., it tries to prove that the certificate was not revoked although it is contained in the CRT, the VA had to construct a different hash tree with root element H_7. One can easily prove that such an adversary could be used to break the collision-resistance of the hash function. The proof is quite similar to the proof of Merkle Hash Trees and is left as an exercise.