Lecture Notes on
Bootstrapping Trust

15-316: Software Foundations of Security & Privacy
Matt Fredrikson[*]

Lecture 18
November 14, 2024

1 Introduction

When we discussed authorization logic, we briefly touched on the topic of trust. More specifically, authorization logic is distinguished from other logics that we have looked at by the A **says** P construct that represents the fact that principal A affirms proposition P. We saw how to encode *decentralized* security policies for systems using **says**, so that various entities can define their own access control rules. The system components that must ultimately either grant or deny access to resources can decide which principals to trust, and subsequently implement a policy by only accepting **says** proclamations from trusted principals.

We illustrated this using the Grey system Bauer et al. [2005], and now we will see a different example that emphasizes the importance of organized trust when making policy decisions. You may be familiar with the eduroam service, which provides members of participating academic institutions with wireless network access when they visit another institution. For example, because both CMU and Pitt are members of eduroam, when you visit the Pitt campus you can join the wireless SSID `eduroam`, and provide your Andrew ID and password to use the internet. The same is true at thousands of other institutions across the world that subscribe to this service.

If you stop to think about it, this is somewhat remarkable given the scale and disparity in geography and governance among the institutions. How does Pitt know that you have entered the valid credentials for your Andrew account, which is managed by CMU? A naive solution might be to distribute the credentials of *all* users at eduroam institutions to all of the institutions. Obviously this will not scale, so perhaps a decentralized authorization policy is called for. First of all, the service

[*]with edits by Giselle Reis, Ryan Riley, and Frank Pfenning

eduroam can delegate the responsibility of deciding who is currently a student to the various institutions, for example as shown below. We use er to denote the eduroam principal.

$$\begin{array}{l} er\ \textbf{says}\ \forall x.\,(cmu\ \textbf{says}\ \mathsf{isStudent}(x)) \to \mathsf{isStudent}(x) \\ er\ \textbf{says}\ \forall x.\,(pitt\ \textbf{says}\ \mathsf{isStudent}(x)) \to \mathsf{isStudent}(x) \end{array}$$

Then the main policy governing access to the eduroam wireless network allows every student to access the network.

$$er\ \textbf{says}\ \forall x.\,\mathsf{isStudent}(x) \to \mathsf{canAccess}(x)$$

The wireless access points responsible for providing the service use this policy as assumptions Γ construct or check a proof of the sequent below when student $derek$ attempts to use the service.

$$\Gamma \vdash er\ \textbf{says}\ \mathsf{canAccess}(derek)$$

For example, suppose that $derek$ attempts to do so. Rather than writing out the proof in the sequent calculus, we present it in the form of a proof term. In the proof-carrying authorization architecture, that is what would be communicated to the access point. More conventionally, it provides a logical justification for granting access but remains implicit in the code implementing access control for the network.

$$\begin{array}{lcl} c_1 & : & er\ \textbf{says}\ \forall x.\,(cmu\ \textbf{says}\ \mathsf{isStudent}(x)) \to \mathsf{isStudent}(x) \\ c_2 & : & er\ \textbf{says}\ \forall x.\,\mathsf{isStudent}(x) \to \mathsf{canAccess}(x) \\ c_3 & : & cmu\ \textbf{says}\ \mathsf{isStudent}(derek) \\ \vdash & & \\ ?M & : & er\ \textbf{says}\ \mathsf{canAccess}(derek) \end{array}$$

$$\begin{array}{l} ?M = \{ \\ \quad \textbf{let}\ \{x_1\}_{er} = c_1\ \textbf{in} \\ \quad \textbf{let}\ \{x_2\}_{er} = c_2\ \textbf{in} \\ \quad \textbf{let}\ x_4 = x_1\ derek\ c_3\ \textbf{in} \\ \quad x_2\ derek\ x_4 \\ \}_{er} \end{array}$$

You may want to refresh your understanding of the details from the notes to Lecture 17.

Because the derivation above will be about the same for any user, except for the details of institution and user names, the endpoint need not recompute or check it for each login. Access boils down to a trusted institution's endorsement that the user is legitimate and eligible for the service, so instead perhaps users' devices can just send evidence of the endorsement directly, or the endpoint can obtain it by some other means.

But how can the access point be sure that this evidence is authentic? We hinted in the last lecture that digital signatures utilizing cryptography are a common solution. But the cryptographic techniques that enable digital signatures require *keys*, which are either secrets distributed among trustworthy parties, or public objects that can be reliably associated with individuals or organizations.

For example, cmu could sign and date a certificate stating that $derek$ is currently a student, perhaps with a timeout to account for Bob's expected graduation date. It would do so using its *private key*, known only to CMU, and the access point would verify it by checking the signature against cmu's *public key*. How does the access point know that it is using the correct public key? What if someone tricked it into using a different public key, associated with an attacker's private key, so that it would trust statements signed by the attacker as coming from cmu?

We will address this topic today, looking more closely at digital certificates and *Public Key Infrastructure* (PKI), which is a distributed mechanism for managing the trust needed to solve the problems introduced in this example.

2 Digital Certificates & Certificate Authorities

For the rest of this lecture, we will assume that all principals A have a secret key sk/A and a public key pk/A. Suppose in the context of the running example from the previous section that the eduroam principal generates the public/private key pairs for all participating institutions.

Digital signatures. One of the main applications of public/secret key pairs is to *digital signatures*, which we have referenced informally before. A digital signature scheme consists of three algorithms, for generating keys, signing messages, and verifying signatures, respectively. We will always assume that public/secret key pairs have been generated correctly by some existing means, so we will not spend any time discussing the key generation algorithm. It is useful however to look a bit more closely at the latter two algorithms, $\mathsf{sign}_{sk/A}(m)$ and $\mathsf{verify}_{pk/A}(s, m)$, to understand how signatures are used to establish trust.

The signing algorithm $\mathsf{sign}_{sk/A}(m)$ takes as input a secret key sk/A for a principal A in addition to a message m, and outputs a signature s. The verification algorithm $\mathsf{verify}_{pk/A}(s, m)$ takes as input a public key pk/A for a principal A, a signature s, a message m, and outputs either true or false. It should be true *if and only if* the signature was produced by calling sign with A's secret key sk/A, i.e., $s = \mathsf{sign}_{sk/A}(m)$ for some m. Otherwise, $\mathsf{verify}_{pk/A}(s, m) = \mathsf{false}$. So in particular $\mathsf{verify}_{pk/A}(s, m)$ will return false if s is a signature created with the secret key of some other principal $B \neq A$, or more formally $\mathsf{verify}_{pk/A}(\mathsf{sign}_{sk/B}(m), m) = \mathsf{false}$ for all m. This is summarized below.

$$\mathsf{verify}_{pk/A}(s, m) = \begin{cases} \mathsf{true} & \text{if } s = \mathsf{sign}_{sk/A}(m) \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Technically speaking, the behavior specified above is only required to hold with overwhelming probability over the keys produced by the generator [Katz and Lindell, 2014]. For our purposes however, it is fine to think of it as holding unconditionally.

The essential property established here is *unforgeability*. As long as sk/A remains a secret, and the only individual who knows the value of sk/A is A, then the only messages that $\mathsf{verify}_{pk/A}(\cdot)$ will return true on are those that A actually signed with sk/A. Of course, if one wanted to forge a message with A's signature, they could attempt to guess sk/A, which is why it is important that secret keys be chosen completely randomly from a very large space of possibilities. It is also important that the outputs of $\mathsf{sign}_{sk/A}(\cdot)$ reveal no information about sk/A that can help one guess the secret key with greater probability. We will assume that all of these facts hold for the secret keys and digital signatures used for the rest of the lecture, and we will also assume that if sk/A was generated by someone other than A (e.g., er in our running example), then they are trusted not to sign messages on A's behalf.

Certificates. Because er knows for a fact that CMU's public key pk/cmu is associated with the correct principal, it can generate a *certificate* that asserts this fact with its signature.

$$\mathsf{cert}_{er\to cmu}(pk/cmu) \equiv \mathsf{sign}_{sk/er}(\mathsf{isKey}(cmu, pk/cmu))$$

The predicate $\mathsf{isKey}(cmu, pk/cmu)$ denotes the fact that the public key pk/cmu belongs to, or is uniquely associated with, the principal cmu. er signs with its secret key sk/er to authenticate the certificate, as no other principals should have knowledge of sk/er and so only er itself could have produced the certificate.

Now if cmu wants to convince one of the access points that $derek$ is in fact a student, it can use $\mathsf{cert}_{er\to cmu}$ as part of a sequence of messages:

$$pk/cmu, \mathsf{cert}_{er\to cmu}(pk/cmu), \mathsf{sign}_{sk/cmu}(\mathsf{isStudent}(derek))$$

As long as the access point has eduroam's public key pk/er, then it will be able to verify that $\mathsf{cert}_{er\to cmu}(pk/cmu)$ is indeed signed by er, and so pk/cmu must really belong to cmu, and then use pk/cmu to verify that cmu signed $\mathsf{isStudent}(derek)$. cmu can send this information to the access point over an insecure channel, and the access point will still be able to trust the final conclusion.

Certificate authorities. Certificates enable the extension of trust to new principals from pre-existing trust relationships. In our running example, er is trusted by all access points to issue certificates for the public keys of other principals. In general, parties endowed with this sort of trust are called *certificate authorities* (CAs). The job of a CA is to issue digital certificates that associate principals with public keys, so in our example the CA is er.

The CA uses their own public/secret key pair to issue certificates, so those who wish to verify certificates issued by a particular CA need a reliable and secure way of obtaining the CA's public key. We will discuss several alternatives for achieving this in the next section, but for now it is fine to assume that all principals are in possession of the correct public key for the CA.

2.1 Formalizing certificates and trust

Now that we have seen how signatures and certificates are used to extend trust relationships, let us think about how to incorporate this into our reasoning about authorization. Specifically, we will formalize policies that utilize signatures, certificates, and trust in the CA so that these elements can be used with existing policies written in authorization logic. We will encapsulate this in a set of policies that can supplement the assumptions used in a proof, but one could alternatively incorporate these principles into axioms in the logic and devise corresponding proof rules [Bauer, 2003].

The first way in which we might want to use signatures is to introduce affirmations. Namely, if we are in possession of a proposition P signed with sk/A, and we know that sk/A is the secret key of A, then we can conclude that A **says** P. We will label this policy c_1 as formalized by the axiom c_1.

$$c_1 : \forall x. \forall pk. \, \mathsf{isKey}(x, pk) \rightarrow \mathsf{sign}_{sk/x}(P) \rightarrow x \ \textbf{says} \ P$$

There is a small notational issue: previously, $\mathsf{sign}_{sk/x}(P)$ was defined as a function returning a signature (typically a string or a large number). Here we are using it as a proposition. This proposition should contain the signature, but also the proposition that was signed. So, more properly, it might be written as $\mathsf{signed}(x, s, P)$ where $s = \mathsf{sign}_{sk/x}(P)$. This is more difficult to read, so we will stick with the shorter $\mathsf{sign}_{sk/x}(P)$.

The assumption that makes c_1 reasonable is that if a principal is willing to sign something, then they are prepared to state it as well. In the base authorization logic, if our proof goal is an affirmation, then we had no choice but to apply $\textbf{says}R$ and subsequently prove an affirmation judgement. It is not difficult to see that c_1 gives us an alternative way of proving such goals, namely that whenever we allow the axiom c_1 then we can prove A **says** P by proving $\mathsf{sign}_{sk/A}(P)$ and $\mathsf{isKey}(A, pk/A)$ from our assumptions.

There is a secondary issue here, namely that c_1 can not be a proper antecedent since it references an *arbitrary* proposition P. Quantifying over all propositions is dicey (and in any case not allowed in our authorization logic). One way we could resolve that is to specialize the axioms to a certain class of propositions. Another is to turn it into an inference rule, the way we have with the axioms of dynamic logic. In that case we would obtain:

$$\frac{\Gamma \vdash \mathsf{isKey}(A, pk/A) \quad \Gamma \vdash \mathsf{sign}_{sk/A}(P)}{\Gamma \vdash A \ \mathbf{says}\ P}\ \mathbf{says/sign}$$

The second premise here uses the special sign predicate which we can prove by algorithmically verifying the signature.

$$\frac{\mathsf{verify}_{pk/A}(\mathsf{sign}_{sk/A}(P), P) = \mathsf{true}}{\Gamma \vdash \mathsf{sign}_{sk/A}(P)}\ \mathbf{verify}$$

This role can only be properly understood with the remark regarding our abuse of notation above: the formula $\mathsf{sign}_{\mathsf{sk/A}}(P)$ should actually contain the signature string s and either the public key pk/A or the principal A.

Now that we have a way of converting signatures into affirmations, we want to apply this towards formalizing the trust extension principle behind certificate authorities and the certificates that they issue. Recall that the trust placed in CAs is that they will sign messages that reliably tell us which principals are bound to particular public keys. So if we know that A is a certificate authority, and we have a certificate $\mathsf{cert}_{A \to B}$, then this principle lets us conclude that pk/B belongs to B. We formalize this as follows:

$$c_2 : \forall x.\, \forall y.\, \forall pk.\, \mathsf{isCA}(x) \to x\ \mathbf{says}\ \mathsf{isKey}(y, pk) \to \mathsf{isKey}(y, pk)$$

Just as the **says/sign** rule simplifies the use of c_1 in proofs, the **cert** rule does so for c_2.

$$\frac{\Gamma \vdash \mathsf{isCA}(A) \quad \Gamma \vdash A\ \mathbf{says}\ \mathsf{isKey}(B, pk)}{\Gamma \vdash \mathsf{isKey}(B, pk)}\ \mathbf{cert}$$

2.2 Example Revisited

Now that we understand how certificate authorities, digital signatures, and certificates work, let us return to the example from the beginning and sketch how to use these elements to prove an affirmation using digital signatures. We pick out as a subgoal the proof that cmu **says** $\mathsf{isStudent}(derek)$. We previously claimed that the message sequence

$$pk/cmu, \mathsf{cert}_{er \to cmu}(pk/cmu), \mathsf{sign}_{sk/cmu}(\mathsf{isStudent}(derek))$$

should convince the verifier that cmu **says** $\mathsf{isStudent}(derek)$. Logically, we express this message sequence, together with our policy and prior knowledge about the er as the following sequent:

$c_1 : \forall x. \forall pk. \, \mathsf{isKey}(x, pk) \rightarrow \mathsf{sign}_{sk/x}(P) \rightarrow x \textbf{ says } P \quad \text{(for all } P\text{)}$
$c_2 : \forall x. \forall y. \forall pk. \, \mathsf{isCA}(x) \rightarrow x \textbf{ says } \mathsf{isKey}(y, pk) \rightarrow \mathsf{isKey}(y, pk)$

$x_3 : \mathsf{sign}_{sk/er}(\mathsf{isKey}(cmu, pk/cmu))$
$x_4 : \mathsf{sign}_{sk/cmu}(\mathsf{isStudent}(derek))$
$x_5 : \mathsf{isCA}(er)$
$x_6 : \mathsf{isKey}(er, pk/er)$
$\vdash$
$?N : cmu \textbf{ says } \mathsf{isStudent}(derek)$

We can define $?N$ as:

$\textbf{let } x_7 = c_1 \; er \; x_6 \; x_3 : er \textbf{ says } \mathsf{isKey}(cmu, pk/cmu) \textbf{ in } \text{(for } P = \mathsf{isKey}(cmu, pk/cmu)\text{)}$
$\textbf{let } x_8 = c_2 \; er \; cmu \; pk/cmu \; x_5 \; x_7 : \mathsf{isKey}(cmu, pk/cmu) \textbf{ in}$
$\textbf{let } x_9 = c_1 \; cmu \; x_8 \; x_4 : cmu \textbf{ says } \mathsf{isStudent}(derek) \textbf{ in} \quad \text{(for } P = \mathsf{isStudent}(derek)\text{)}$
x_9

This completes the proof. Now the access point can conclude that cmu vouches for $derek$ and so according to the original eduroam policy that he is allowed to use the network. Notice how the access point is able to draw this conclusion by trusting only the CA, er in the beginning, and not cmu or $derek$. From this initial seed of trust it was able to "bootstrap" the additional trust assumptions that it needed to apply the authorization policy. This idea of bootstrapping trust from a few entities to many through CA designations is a key takeaway of this lecture, and one that is widely used in practice to enforce authorization on large-scale distributed systems.

2.3 Failure modes

Can principals always rely on certificates and trust relationships to establish authenticity of messages? There are a few situations that the access point needs to worry about, and they have to do with the assumption that *private keys are only known to their respective principals*. If this assumption ever fails, then problems can crop up in a few places in our running example.

The first case where the assumption can fail is for cmu. Supposing that cmu's private key used for signing messages (e.g., $\mathsf{sign}_{sk/cmu}(\mathsf{isStudent}(derek))$) becomes compromised and leaks to an untrusted individual who is not authorized to make statements on behalf of cmu. Then this person can sign messages of their choosing and have others who believe that pk/cmu belongs to cmu believe them with reasonable evidence. In the context of the example, that individual could sign things that are patently false, such as $\mathsf{isStudent}(beyonce)$, and the access point would believe that the messages originated from cmu. Recalling that the assertion **says** $cmu(\mathsf{isStudent}(x))$ is the only thing one needs to establish to access the network, this obviously renders the access control ineffective.

The other case corresponding to compromise of eduroam's secret key has similar consequences when considered in the context of our example. If an attacker is in possession of eduroam's secret key, then they gain the ability to generate certificates that look like they came authentically from eduroam. So rather than using cmu's secret key directly, this attacker would generate a separate public/secret key pair $\langle pk/\star, sk/\star\rangle$, and use er's key to certify that $pk/\star$ belongs to cmu.

$$\mathsf{cert}_{er\to cmu}(pk/\star) \equiv \mathsf{sign}_{sk/er}(\mathsf{isKey}(cmu, pk/\star))$$

They could then convince the access point to allow any principal of their choosing to use the network, sending the messages such as

$$pk/\star, \mathsf{cert}_{er\to cmu}(pk/\star), \mathsf{sign}_{sk/\star}(\mathsf{isStudent}(beyonce))$$

As in the previous case, compromise of the secret key sk/er renders the access control system pointless.

But outside the narrow context of our example, compromise of signing keys belonging to parties that are widely trusted to certify identities and establish policies is extremely serious. Without additional measures in place that we will discuss later, it gives one the ability to fabricate and steal the identities of arbitrary individuals. This can have dire consequences.

For example, suppose that A and B wish to communicate using their public and private keys. They trust certificates signed by C, and so if A wishes to send B an encrypted message, then B will first send A their public key pk/B and a certificate issued by C that attests to the validity of that public key. Then A can encrypt the message using B's public key, $\mathsf{enc}_{pk/B}(\ldots)$, and B will be able to decrypt with their secret key sk/B.

$$\begin{tikzcd}[column sep=huge]
A & B \\
{} & {} \arrow[l, "{pk/B, \mathsf{sign}_{sk/C}(\mathsf{isKey}(B, pk/B))}"'] \\
{} \arrow[r, "{\mathsf{enc}_{pk/B}(\text{“Hello!”})}"'] & {}
\end{tikzcd}$$

Now suppose that a malicious party M has obtained C's secret signing key. Then if M is able to intercept all messages passed between A and B, they can read the encrypted messages intended for B as well as make changes to them. When B sends A its public key and cert signed by C, then M uses C's signing key to certify a chosen public key $pk/B^\star$ (with corresponding secret key $sk/B^\star$ known to M), and forward $pk/B^\star$ to A with certification instead of pk/B. A will believe that $pk/B^\star$ is B's public key because it came with a certificated signed by trusted principal C, and use it to encrypt messages to B as shown below.

$$\begin{tikzcd}
A & M & B \\
{} \arrow[r, leftarrow, "pk/B^*, \mathsf{sign}_{sk/C}(\mathsf{isKey}(B, pk/B^*))"] & {} \arrow[r, leftarrow, "pk/B, \mathsf{sign}_{sk/C}(\mathsf{isKey}(B, pk/B))"] & {} \\
{} \arrow[r, "\mathsf{enc}_{pk/B^*}(\text{“Hello!”})"'] & {} \arrow[r, "\mathsf{enc}_{pk/B}(\text{“Goodbye!”})"'] & {}
\end{tikzcd}$$

Of course, because M knows the corresponding secret key $sk/B^\star$, it can decrypt and inspect the private messages A sent to B. It can then choose to either re-encrypt the original message with pk/B, or one of its choosing. This is called a *Man-in-the-Middle* (MitM) attack, as the attacker literally situates in between two parties who believe they are communicating over a secure channel.

3 Public key infrastructure

So far we have glossed over the details of how certificate authorities are assigned and managed. In the eduroam example, we assumed that access points know the correct pk/er because they are provisioned expressly for the service, and come preloaded with the necessary data. But certificates are used in all sorts of applications, and it may not always be possible to transmit the CA's key in such a way. How do principals come to trust a CA, and how does the CA know that pk/A actually belongs to A in cases where it does not generate the key? Answers to these questions entail defining a *Public Key Infrastructure* (PKI), and there are several alternatives for doing so.

3.1 Centralized CA

The most basic type of PKI consists of a single certificate authority who is trusted by all principals to issue certificates for everyone's public keys. Anyone who wants to use the PKI to establish trust in other principals must obtain a secure copy of the CA's public keys, and if they fail to accomplish this, then they will be unable to verify legitimate certificates issued by the true CA, and may instead end up "verifying" forged certificates issued by attackers. Protecting against this possibility is typically accomplished by obtaining a copy of the CA's key through physical contact, i.e., visiting the CA's offices and obtaining a file whose contents can be compared against a known checksum. Likewise, to obtain a certificate principals must usually present physical evidence of who they are, and that the keys they wish to have signed actually belong to them. Although the details of how this is done vary between CAs, the basic process must be transparent and rigorous enough so that others trust the CA's certs.

Another popular form of distribution for this model is to bundle public keys for widely-known CAs with popular software. This is done with browsers and operating systems, which typically implement a key store that is pre-loaded with CA keys that can be automatically verified as needed for validation. However, this approach is not without its risks, as users are often tricked into downloading corrupted versions of software that may have additional keys not associated with real CAs pre-loaded into the store. In this event, all of the failure modes discussed in the previous section are possible and likely, which is why it is important to always verify checksums for software that needs to interact with PKI.

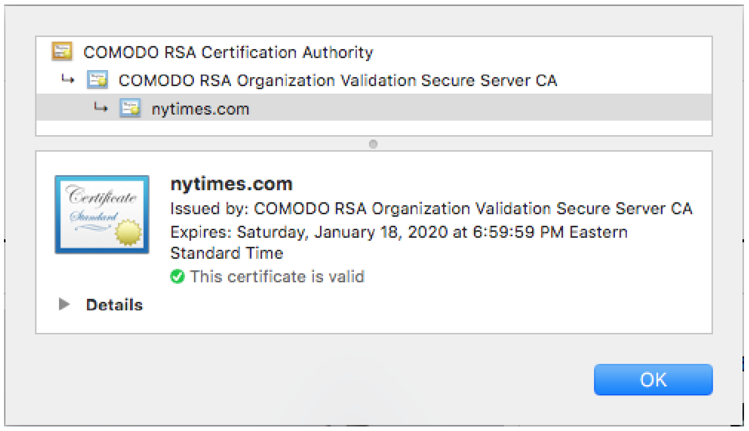
This type of CA is usually a company that charges a fee to issue certificates, or department within an organization tasked with overseeing security. Because issuing certificates is a lucrative business model, in practice there are many "centralized" CAs that exist, and principals are free to choose whichever one they like when purchasing certificates for their keys. In one important sense this makes the overall PKI, in which users can choose which CAs to use and trust, less brittle to compromise of any one CA's signing key. In fact, it is considered good practice by some to obtain multiple certificates for the same key, so that if one CA is corrupted then others still have reason to trust the authenticity of the key. However, most browsers and operating systems that come pre-loaded with CA keys are configured to trust all of them equally, so really the entire PKI is only as trustworthy as the least-trustworthy CA. Ultimately, the responsibility is placed on end-users to configure their settings in response to corrupt or compromised CAs as such information becomes available. This is an unfortunate reality as most users are not equipped to make such decisions, and fixing it remains an open problem.

3.2 Delegated trust and hierarchical CAs

The reality of key compromise and the wide geographical reach of large certificate authorities has led to the emergence of an alternative hierarchical PKI. This model extends the centralized approach, and still makes use of the key distribution and principal verification strategies used by the centralized model. But now, the primary "root" CA *delegates* the ability to issue certificates to a number of subsidiary or "second-level " CAs. Certificates issued by second-level CAs then come with root-issued CAs themselves, thus forming a "certificate chain" that can be verified in sequence until reaching a trusted root CA with a known public key.

We can formalize this delegation policy using authorization logic. All that the CA needs to do is sign propositions that denote which principals they trust to sign on their behalf, e.g. with a predicate $\mathsf{trusts}(\ldots)$. Then the subsidiary CA can attach the policy signed by the root CA to any certificate that it issues using its delegation privilege.

$$\begin{aligned}CA \ \mathbf{says}\ (\forall x.\, \forall y.\, \forall pk.\ CA \ \mathbf{says}\ &\mathsf{trusts}(x) \\ &\to x \ \mathbf{says}\ \mathsf{isKey}(y, pk) \to \mathsf{isKey}(y, pk)\end{aligned}$$


COMODO RSA Certification Authority
↳ COMODO RSA Organization Validation Secure Server CA
↳ nytimes.com

nytimes.com
Issued by: COMODO RSA Organization Validation Secure Server CA
Expires: Saturday, January 18, 2020 at 6:59:59 PM Eastern Standard Time
✓ This certificate is valid

Figure 1: Example certificate chain used to authenticate the secure `nytimes.com` website, as displayed in the Chrome browser.

Checking that the rules described earlier allow others to make effective use of subsidiary-issued certificates is left as an exercise.

An example of this model in action is shown in Figure 1, which is the current certificate provided when visiting `https://nytimes.com`. The root CA in this case is *COMODO RSA Certification Authority* (we will call this A), and the second-level CA is *COMODO RSA Organization Validation Secure Server CA* (we will call this B), which is the principal who signed the public key of `nytimes.com`. The browser verifies that the certificate for `nytimes.com` was signed by B, and that the certificate for B was signed by the root CA A. In addition, the browser will check that the root-signed certificate for B's public key is authorized to sign certificates itself; this is a special "extension" field supported by the standard (X.509 [IETF, a]) for digital certificates. This special designation essentially says that B is trusted by A to issue additional certificates on behalf of A, and that those certificates should be treated as though they were issued by A itself.

The ability to delegate certification authority addresses many of the practical hurdles in the centralized CA model. Certificate authorities need to shoulder several burdens: ensuring the secrecy of the signing key, ensuring that the public key is readily available for verification, and vetting clients who wish to obtain certificates. Splitting these responsibilities among several subsidiaries makes good logistical sense. However, it also means that instead of just one signing key, there are now several that must be kept secret. The root CA must also ensure the integrity of subsidiary CAs, as they have the ability to issue certificates on behalf of the root CA, and so the trustworthiness of all related CAs is defined by the least trustworthy subsidiary. In short, while the hierarchical model solves some problems, it introduces several others.

Given that this model is in use on the Internet, exactly how many different organizations can function as CAs and sign keys that your web browser will trust? It is difficult to be sure, because the distributed nature of delegation means that there is no central repository listing all CAs and who has delegated to whom. A study done by the Electronic Frontier Foundation's SSL Observatory in 2010 found 651 distinct organizations functioning as CAs. That number is likely even higher today.

3.3 Web of trust

An alternative PKI model to the hierarchical trust model is known as *Web of Trust* (WOT). While the hierarchical model is widely deployed in operating systems and web browsers, WOT has been in use for several decades particularly in the context of the *Pretty Good Privacy* (PGP) [Garfinkel, 1995] project. In WOT, trust is completely decentralized and users are responsible for making their own decisions about which certificates to trust. Likewise, every user is able to issue certificates as they wish, and distribute them at-will to others.

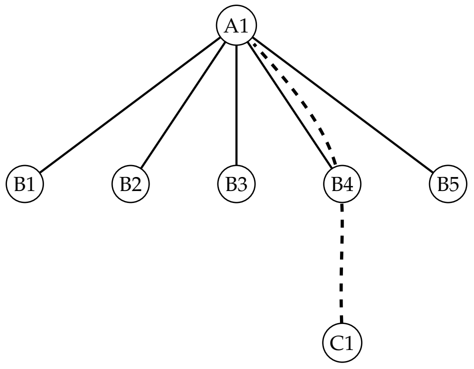
To get an idea of how this might work, consider a scenario where $giselle$ wishes to send $matt$ an email encrypted with her secret key. She sends her public key, along with certificates $\mathsf{cert}_{mike \rightarrow giselle}(pk_1)$ signed by the CMUQ dean Michael Trick and $\mathsf{cert}_{ryan \rightarrow giselle}(pk_2)$ signed by $ryan$. Suppose that $matt$ does not know $mike$ (because $matt$ has never visited CMUQ). Suppose that he does know $ryan$, because they have corresponded previously and so $matt$ has already established the authenticity of $ryan$'s public key. He can thus verify the first cert $\mathsf{cert}_{ryan \rightarrow giselle}(pk_1)$, and authenticate $giselle$'s public key prior to decrypting.

The main advantage of WOT over the prior two models is the distribution of potential failure. There are no concentrated points of failure in the event of compromise, and everyone is incentivized to proactively authenticate the public keys of anyone they communicate with. Over time, this tends to build redundancy into the system so that if any one user's signing key becomes compromised then anyone who may need to use a cert issued by them will still have several options available.

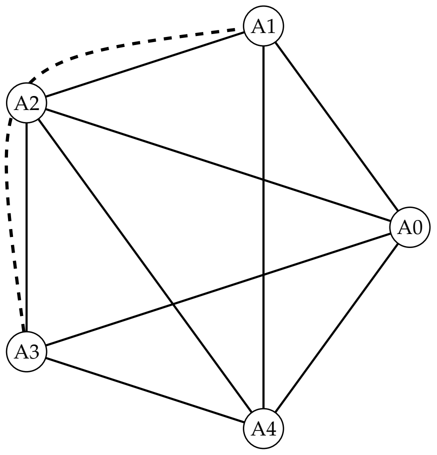
The main drawback is scalability and usability. While WOT remains in use in the context of encrypted email, it has not become an established alternative for other applications as it is difficult and time-consuming to develop a robust network of trust relationships. Additionally, users who are not familiar with public key cryptography face hurdles in being tasked with maintaining a secure and extensive set of trust relationships and certificates, and it is not at all clear that this approach is usable outside of the relatively homogeneous group of PGP devotees.

3.4 Dealing with certificate compromise

So far we have discussed the possible consequences of key compromise, and weighed the potential ramifications of several PKI models on this outcome. But what happens when a signing key becomes compromised? This poses a significant chal-



(a)



(b)

Figure 2: Hierarchical (a) and Web of Trust (b) PKI models, where solid lines correspond to existing trust relationships and dashed curves to the certificate chains that must be verified to build new ones. In the hierarchical model, A1 is the root CA, and B1-B5 are the second-level subsidiary CAs. The second-level CAs issue most certificates, so if one wants to verify C1's certificate then they need to first check that B4 signed C1's key, and then verify that A1 signed B4's key. In Web of Trust, all parties occupy a flat hierarchy, and verify certificates using previously-verified keys. If A3 wishes to authenticate A1's key, A3 can ask for a certificate signed by A3, who is a common point of trust between the two parties.

lenge, as continued trust in signatures issued with that key cripples the security of applications that rely on it. The CA needs to disavow, or *revoke*, the compromised key immediately while ensuring that users are aware that they should no longer trust the old one. There are several approaches for doing this, none of them entirely satisfactory. At present, this remains an open and active research question.

Expiration. Nearly all certificates in use today were issued with an expiration date, as shown in Figure 1. This facilitates a "default" mode of protection against key compromise, as once the expiration date passes verifiers will no longer trust the certificate. However, expiration alone is not sufficient to fully address the problem, as there is an untenable conflict between scalability and the burden on CAs to continually issue and distribute new certificates, and the "window of vulnerability" between compromise and the certificate's expiration date. In other words, it is not considered feasible to set short certificate lifetimes of, say, one day to one week, because if this were common practice then there would be no way for CAs to keep up with the logistical requirements needed to constantly re-issue new certificates. Typically, CA-issued certificates have lifetimes that last several years, and this leaves the parties in question with a potentially large time span in which their operations are affected by compromise. One notable exception to this is Let's Encrypt, which has automated the entire process of issuing certificates and so is logistically able to support three month lifetimes for certificates.

Certificate revocation lists. The most common way of handling this problem is for the CA to maintain a *certificate revocation list* (CRL). Each certificate is given a unique serial number, and if the key becomes compromised then the CA is notified that the certificate with the corresponding serial number should be revoked. The CA distributes an updated CRL each day, and verifiers are responsible for cross-referencing the list when checking a certificate.

Because new CRLs must be obtained by users regularly, this solution imposes a significant burden on the PKI. Whereas before communications could take place "offline" without needing to communicate with services exclusive to PKI, this is no longer the case. If the CRL server goes offline, either incidentally or as the result of an explicit attack, then users can no longer verify certificates without running the risk of accepting one signed by a compromised key. Additionally, CRLs tend to grow quite large over time, and this leads to non-trivial bandwidth costs for ISPs and end-users, particularly those who operate mobile devices. While proposals for incremental CRLs exist ([IETF, a], Section 5.2.4), they are not widely implemented.

Online Certificate Status Protocol (OCSP). An alternative to certificate revocation lists is OCSP, which is an active protocol in which parties "pull" information about certificate status rather than having CAs "push" the information routinely [IETF, b]. The details of the protocol are not immediately relevant to our discussion, but it does offer some interesting tradeoffs to CRLs. The problem that OCSP alleviates is the transmission of large CRLs to end users. Because typical OCSP data transfers occur in response to specific certificate transactions, the amount of data in them is significantly smaller and therefore easier to parse and manage on resource-constrained devices. However, where connectivity or latency are issues, OCSP may become more burdensome than if an up-to-date CRL were stored on the device.

OCSP has also raised concern from privacy advocates. The on-demand "pull" nature of the protocol essentially requires users to tell a central third party whose certificates they would like to validate. Because much of the web now supports HTTPS, which requires certificate validation, this means that visiting a secure website from a browser that uses OCSP (this includes Internet Explorer, Mozilla, Safari, and Opera, but not Chrome) results in the OCSP server learning that the user visited that website. To make matters worse, the OCSP proposed standard does not mandate encryption by default, so third parties sitting between the user and the OCSP server may also be able to snoop on these requests. For these reasons, the support for OCSP seems to be waning [Aas, 2024].

Certificate pinning. A fairly recent practice called *certificate pinning* addresses the possibility of CA key compromise. Because there are dozens of root CAs that browsers are configured to trust by default, if any one of these CAs becomes compromised then the attacker can issue certificates as that CA for *any* user or domain. So suppose that `cmu.edu` contracts with only one CA for certificate issuance, $trustedCA$. Now a new CA, $discountCA$, enters the marketplace and quickly becomes compromised thanks to their lax security standards. If browsers and operating systems are already configured to trust $discountCA$, then the party who compromised their key can now issue certificates for `cmu.edu`, even though `cmu.edu` never used the services of $discountCA$!

Certificate pinning addresses this by allowing parties to "pin" a set of trusted CAs, so that verifiers will only trust the public keys of pinned CAs chosen by `cmu.edu`; in this case, `cmu.edu` would only pin $trustedCA$. Certificate pinning is now common practice when configuring HTTPS websites, and is supported by all major browsers. One drawback to certificate pinning is that it can obviate legitimate network security tools that essentially use man-in-the-middle attacks to scan encrypted network traffic for malicious content. Another drawback is that if the CA becomes compromised, then nobody will be able to verify certificates pinned to that CA until the pin expiration date arrives.

References

Josh Aas. Intent to end OCSP service. `https://letsencrypt.org/2024/07/23/replacing-ocsp-with-crls/`, July 2024.

Lujo Bauer. *Access Control for the Web via Proof-Carrying Authorization*. PhD thesis, Princeton University, November 2003.

Lujo Bauer, Scott Garriss, Jonathan M. McCune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of the 8th Information Security Conference (ISC'05)*, pages 431–445, Singapore, September 2005. Springer Verlag LNCS 3650.

Simson L. Garfinkel. *PGP - Pretty Good Privacy: Encryption for Everyone*. O'Reilly, 2nd edition edition, 1995.

IETF. RFC 6960: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) profile. `https://tools.ietf.org/html/rfc6960`, a.

IETF. RFC 5280: Internet X.509 Public Key Infrastructure Online Certificate Status Protocol – OCSP. `https://tools.ietf.org/html/rfc5280`, b.

Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2nd edition, 2014.