CHAPTER

18

The Dream of PKI

In this chapter we will give the standard presentation of what a PKI is, and how it solves the key management problem. It is important to understand this first. In the next chapter we'll talk about the challenges with PKIs in practice, but for this chapter we'll visit the perfect world where a PKI solves all your problems.

18.1 A Very Short PKI Overview

A PKI is a *Public-Key Infrastructure*. It is an infrastructure that allows you to recognize which public key belongs to whom. The classical description is as follows.

There is a central authority that is called the *Certificate Authority*, or CA for short. The CA has a public/private key pair (e.g., an RSA key pair) and publishes the public key. We will assume that everybody knows the CA's public key. As this key remains the same over long periods of time, this is easy to accomplish.

To join the PKI, Alice generates her own public/private key pair. She keeps the private key secret, and takes the public key PK_A to the CA and says: ''Hi, I'm Alice and PK_A is my public key.'' The CA verifies that Alice is who she says she is and then signs a digital statement that states something like ''Key PK_A belongs to Alice.'' This signed statement is called the *certificate*. It certifies that the key belongs to Alice.

If Alice now wants to communicate with Bob, she can send him her public key and the certificate. Bob has the CA's public key, so he can verify the

signature on the certificate. As long as Bob trusts the CA, he also trusts that PK_A actually belongs to Alice.

Using the same procedures, Bob gets his public key certified by the CA, and sends his public key and certificate to Alice. They now know each other's public key. These keys in turn can be used to run the key negotiation protocol to establish a session key for secure communications.

What is required is a central CA that everybody trusts. Each participant needs to get his or her public key certified, and each participant needs to know the CA's public key. After that, everybody can securely communicate with everybody else.

That sounds simple enough.

18.2 PKI Examples

To make the rest of this chapter easier to understand, we'll first give some examples of how PKIs can be implemented and used.

18.2.1 The Universal PKI

The ultimate dream is a universal PKI. A large organization, like the post office, certifies everybody's public key. The beauty of this is that every person only needs to get a single key certified, as the same key can be used for every application. Because everybody trusts the post office, or whatever other organization becomes the universal CA, everybody can communicate securely with everybody else, and they all live happily ever after.

If our description sounds a bit like a fairy tale, that is because it is. There is no universal PKI, and there never will be.

18.2.2 VPN Access

A more realistic example would be a company that has a VPN (Virtual Private Network) to allow its employees to access the corporate network from home or from their hotel room when they are traveling. The VPN access points must be able to recognize the people who have access and exactly what level of access they have. The IT department of the company acts as the CA and gives every employee a certificate that allows the VPN access points to recognize the employee.

18.2.3 Electronic Banking

A bank wants to allow its customers to perform financial transactions on the bank's website. Properly identifying the customer is vital in this application, as is the ability to produce proof acceptable in court. The bank itself can act as the CA and certify the public keys of its customers.

18.2.4 Refinery Sensors

A refinery complex is very large. Spread out between miles of pipes and access roads are hundreds of sensors that measure things like temperature, flow rate, and pressure. Spoofing sensor data is a very serious attack on the refinery. It might not be too difficult to send false sensor data to the control room, tricking the operators into taking actions that lead to a large explosion. Therefore, it is imperative that the control room get the proper sensor readings. We can use standard authentication techniques to ensure that the sensor data has not been tampered with, but to be sure that the data actually comes from the sensor, some kind of key infrastructure is needed. The company can act as a CA and build a PKI for all the sensors so each sensor can be recognized by the control room.

18.2.5 Credit Card Organization

A credit card organization is a cooperative venture between a few thousand banks spread out all over the world. All of these banks must be able to exchange payments. After all, a user who has a credit card from bank *A* must be able to pay the merchant that banks with bank *B*. Bank *A* will need to settle with bank *B* in some way, and that requires secure communications. A PKI allows all banks to identify each other and perform secure transactions. In this situation, the credit card organization can act as the CA that certifies the keys of each bank.

18.3 Additional Details

In real life, things become somewhat more complicated, so various extensions to the simple PKI scheme are often used.

18.3.1 Multilevel Certificates

In many situations, the CA is split into multiple pieces. For example, the central credit card organization is not going to certify each bank directly. Instead, they will have regional offices to deal with the individual banks. You then get a two-level certificate structure. The central CA signs a certificate on the regional CA's public key that says something like: ''Key PK_X belongs to regional office *X* and is allowed to certify other keys.'' Each regional office can then certify individual bank keys. The certificate on the bank's key consists of two signed messages: the central CA's delegation message that authorizes the regional office's key, and the regional office's certification of the bank's key. This is called the *certificate chain*, and such a chain can be extended to any number of levels.

Such multilevel certificate structures can be very useful. They basically allow the CA functionality to be split into a hierarchy, which is easy to handle for most organizations. Almost all PKI systems have a multilevel structure. One disadvantage of this structure is that the certificates grow larger and require more computations to verify, but this is a relatively small cost in most situations. Another disadvantage is that each extra CA that you add to the system provides another point of attack, and thereby reduces overall system security.

One way to reduce the disadvantage of the large multilevel certificates that we have not seen in practice would be to collapse the certificate hierarchy. To continue with this example, once the bank has its two-level certificate, it could send it to the central CA. The central CA verifies the two-level certificate and replies with a single certificate on the bank's key, using the master CA key. Once the key hierarchy is collapsed like this, the performance cost of adding extra levels to the hierarchy becomes very small. But then again, adding extra layers might not be such a good idea; many-layered hierarchical structures are rarely effective.

You have to be careful when chaining certificates together like this. They add more complexity, and complexity is in general risky. Here is an example. Secure sites on the Internet use a PKI system to allow browsers to identify the correct website. In practice, this system isn't very secure, if only because most users don't verify the name of the website they are using. But a while back, a fatal bug showed up in a library that validates certificates on all Microsoft operating systems. Each element of the certificate chain contains a flag that specifies whether the key it certifies is a CA key or not. CA keys are allowed to certify other keys. Non-CA keys are not allowed to certify other keys. This is an important difference. Unfortunately, the library in question didn't check this flag. So an attacker could buy a certificate for the domain `nastyattacker.com` and use it to sign a certificate for `amazon.com`. Microsoft Internet Explorer used the faulty library. It would accept `nastyattacker.com`'s certification of a fake Amazon key and show the fake website as the real Amazon website. Thus, a worldwide security system that cost a fortune to build was completely outflanked by a simple little bug in a single library. Once the bug was published, a patch was released (it took several tries to fix all the problems), but this remains a good example of a minor bug destroying the security of an entire system.

18.3.2 Expiration

No cryptographic key should be used indefinitely; there is always a risk that the key will be compromised. Regular key changes let you recover from compromise, albeit slowly. A certificate should not be valid forever, either, because both the CA's key and the public key that is being certified expire.

Apart from these cryptographic reasons, expiration is important in keeping information up-to-date. When a certificate expires, a new one will have to be reissued, and this creates an opportunity to update the information in the certificate. A typical expiration interval is somewhere between a few months and a few years.

Almost all certificate systems include an expiration date and time. Nobody should accept the certificate after this date and time. This is why participants in a PKI need a clock.

Many designs include other data in the certificate. Often certificates have a not-valid-before time, in addition to the expiration time. There can be different classes of certificates, certificate serial numbers, date and time of issue, etc. Some of this data is useful, some useless.

The most commonly used format for certificates is X.509 v3, which is overly complicated. See Peter Gutmann's style guide [58] for a discussion of X.509. If you work on a system that doesn't have to be interoperable with other systems, you might strongly consider forgetting about X.509. Of course, X.509 is standardized, and it's hard to fault you for using a standard.

18.3.3 Separate Registration Authority

Sometimes you will see a system with a separate registration authority. The problem is a political one. It is the HR department of a company that decides who is an employee. But the IT department has to run the CA; that is a technical job that they are not going to allow the HR department to do.

There are two good solutions to this. The first one is to use a multilevel certificate structure and let the HR department be its own sub-CA. This automatically provides the necessary flexibility to support multiple sites. The second solution is much like the first one, except that once a user has a two-level certificate, he exchanges it for a one-level certificate at the central CA. This eliminates the overhead of checking a two-level certificate each time it is used, at the cost of adding a simple two-message protocol to the system.

The really bad solution is to add a third party to the cryptographic protocol. The project specifications will talk about the CA and another party that might be called something like the RA (Registration Authority). The CA and RA are treated as completely separate entities, which can add more than 100 pages of documentation to the system. That is bad in itself. Then there is the need to specify the RA–CA interaction. We've even seen three-party protocols in which the RA authorizes the CA to issue a certificate. This is a good example of the problem of imposing user requirements on a technical solution. User requirements only specify the outside behavior of a system. The company needs to have separate functionality for the HR and IT departments. But that does not mean the software has to have different code for the HR and IT departments. In many situations, and certainly in this one, the

two departments can use much of the same functionality, and thus much of the same code. Using a single set of certificate functions leads to a design that is simpler, cheaper, more powerful, and more flexible than one based directly on the original requirements that included both a CA and an RA entity. A two-level CA scheme allows HR and IT to share most of the code and protocols. The differences, in this case, are mostly in the user interface and should be easy to implement. That translates to maybe a few hundred lines of extra code, not a few hundred extra pages of specifications that turn into tens of thousands of lines of code.

18.4 Summary

What we have described is a dream, but a very important dream. PKI is the first and last word on key management for most of our industry. People have been brought up on this dream and see it as something so obvious that it doesn't need stating. To be able to understand them, you must understand the PKI dream, because a lot of what they say is within the context of the dream. And it feels *so* good to think that you have a solution to the key management problem . . . .

18.5 Exercises

Exercise 18.1 Suppose a CA is malicious. What bad things could the CA accomplish?

Exercise 18.2 Assume a universal PKI. Can any security problems arise because of the use of this single PKI across multiple applications?

Exercise 18.3 What policy or organizational challenges might impede or prevent the deployment of a worldwide universal PKI?

Exercise 18.4 In addition to the examples in Sections 18.2.2–18.2.5, give three example scenarios for which a PKI might be viable.