PKI: what were we thinking?

Author: André Mariën
Date: February 28, 2012

Table of contents
PKI: what were we thinking?.............................................................................................. ... 6
Table of contents............................................................................................................. 1
Abstract.......................................................................................................................... 1
Introduction...................................................................................................................... 2
Primary use case: confidential communication ............................................................... 2
Confidential communication without prior contact ......................................................... 2
Public-private key to the rescue ...................................................................................... 2
Getting the public key, securely....................................................................................... 3
The easy way – let the user decide.................................................................................. 3
The seemingly easy way - Hierarchies of keys............................................................... 3
First major flaw: the management of trusted CAs .......................................................... 3
Who do we trust to define trusted CAs? ......................................................................... 3
How are the CA keys distributed? .................................................................................... 4
CAs as in use now are a bad idea anyway ....................................................................... 4
Only use CAs for purposes that make sense................................................................... 5
Trust in a CA implies trust in the operational context .................................................. 5
Second major flaw: undoing certificate issuing .............................................................. 6
Expiration, CRL and OCSP ............................................................................................ 6
Let certificates expire........................................................................................................ 6
Invalidate a certificate – not possible............................................................................... 6
Wait a second … objective??.......................................................................................... 7

Abstract
Public Key Infrastructure has been around since the early days of the web. It has been key to anything serious on the internet: internet banking, shopping, remote access, etc. It has always been on the edge of the Great Breakthrough. Over and over this did not happen and cracks in its foundation keep showing up. A core technology that is in massive use with no real alternative is SSL/TLS. Certificate Authorities are a key enabler for this. The CA concept and operation has always been a concern but it is heating up fast nowadays. The principles behind current PKI operation are questionable and the CAs approach is bankrupt. There is no easy fix so expect this situation to last. We propose one way the situation could be improved.
Introduction

Primary use case: confidential communication
The problem that SSL/TLS is supposed to solve is to establish confidential communication between two parties. To do this, two objectives must be reached: authentication and communication confidentiality. They are linked: it is not possible to set up a confidential connection with a party unless you are sure it is the right party. To put it differently: without authentication of some sort, a man-in-the-middle attack is obvious.

In principle one would expect mutual authentication to be necessary. In practice, one side is ok. Setting up a connection is asymmetric. There is one party setting up a connection, the client. The client may be anonymous: the server has no clue who he is talking to, but the client knows he talks to the right server confidentially.

Once this connection exists, it is possible to authenticate the client using the confidential channel, allowing most any authentication method, to reach mutual authentication. It is possible and part of the SSL protocol to use certificates for client authentication, but this is not in the scope of the current discussion.

Confidential communication without prior contact
It is not too hard to communicate confidential with a server you know and love and were this relationship is reciprocal: you exchange in private the key(s) used to create a confidential communication channel.

The business case for SSL is to be able to communicate in private with any server on the internet. This is hard. Any communication prior to the establishment of the secure connection is obviously insecure. We tend to forget that the first time we looked into this problem is seemed impossible to solve it.

Public-private key to the rescue
Both for the key exchange and the server authentication public-private key encryption can be used. In step one the client must obtain the public key of the server securely. Suppose this is done. We will have to get back to this questionable item later.

The server public and private key can be used for three basic functions. First, the server can sign with its private key to prove it is indeed that server. Second, signed data provide integrity protection for data it sends to the client. Third, the client can use the public key to protect data for anyone that does not have the private key. Normally only the server has this key.

The SSL protocol uses the above three public-private key technology features.
- The server will sign a challenge sent by the client to authenticate.
- The client can send keys or pre-key data protected by the server’s public key.
• The protocol exchanges can be checked for integrity if the server signs the exchange.

This is all well-known and the last few bugs in the protocol have been eliminated recently, we hope.

**Getting the public key, securely**

**The easy way – let the user decide**

The detail to solve is how the client gets the public key of the server securely, not just supposing it is done. The software could just warn the user: this is the public key presented by the server, do you trust it? Under the assumption that chances are very low that this very communication is under attack, it can work. If the user has a secondary means to validate the key, the better. SSH uses this approach. The approach relies on the first connection and the acceptance of the key in that case. It does somehow assume that this first connection is made under secure circumstances, or that the user is sufficiently aware of the risks and uses extra checks to ensure the key is the correct one. In theory this can be ok; in practice it is often ok, because a systematic, lasting attack is uncommon, not because users are careful.

The alternative to safely exchange some keys is the use of hierarchies of keys.

**The seemingly easy way - Hierarchies of keys**

The common web approach is using hierarchies of keys. This reduces the number of keys to obtain in a trusted way. The public server key and an identifier (here: its domain name) are signed by another party, a certificate authority (CA). Suppose one obtained the public key of a CA in a trusted way, the server now presents a public key signed by that CA. The client checks the identity and the signature and if all is well, now has the public key of the server. A single CA can issue as many server certificates as needed, reducing significantly the number of keys to manage, if one trusts the CA.

One is not restricted to one layer: the new keys can itself by signed by a higher level CA and so on. In principle, there could be one CA to lead them all. History and fantasy (LotR) both showed this is a really bad idea. Fortunately, SSL stopped one step earlier.

We have to address two remaining problems: who do we allow to be a CA, or, in other words, who decides on the hierarchies of trust and its members, and, once more, how are the top level keys securely distributed?

**First major flaw: the management of trusted CAs**

**Who do we trust to define trusted CAs?**

This is a very interesting question. There is no official authority at all that publishes an approved or endorsed CA list, or which has final saying over such a list, or that issues requirements or that checks the conformance. In part, this is by design: the internet design is full of distributed responsibility and trust. No single organization would be accepted worldwide as root of trust. Agreement on the requirements for CAs would be very
difficult to obtain, even if one would solve the more fundamental question on who has to agree.

Software vendors producing browsers decide in practice on the CAs that are OK. Let us assume they do a good job for now. But they are not the instances that should be the guardian of worldwide security.

A similar point was also raised in SANS NewsBites Vol. 14 Num. 15, by Murray:

[Editor's Note (Murray): Our reliance on Browser publishers for the enforcement of PKI, a role in which they have only a limited financial interest, is problematic.]

I do not think that “in which they have only a limited financial interest” really matters a lot, in this context, but it does not help for having confidence in the result.

**How are the CA keys distributed?**

The CA hierarchy is often distributed as part of the browser distribution. The software producers decide which CA they will include. They can issue whatever they deem sufficient requirements and they can decide what constitutes sufficient proof of compliance with those requirements.

The security of the software containing CA certificates as a consequence also defines the security of the CA key distribution, again coupling two issues that are not intuitively linked.

Browsers may be pre-installed. Browser downloads are most often HTTP, not HTTPs downloads or even plain FTP. Arguably, if the software package has been tampered with, you have more to worry about than just a CA key store that is corrupt.

**CAs as in use now are a bad idea anyway**

Every CA is born very equally. The certificates it issues can be for any of the major reasons: SSL, client authentication, email, code signing, etc. for any system anywhere in the world. Top level CAs usually do not have but self-imposed restrictions. Secondary CAs have a profile corresponding to the expected use: server certificate signers, email certificate signers, code signers …

Ok, you have options to change it and tune it:

![Certificate purposes](image)
It is not obvious to find how to do it, let alone explain to people when to change some settings and why.

If you trust any CA you trust it completely: if it can sign server certificates, it can sign any server certificate, regardless of the domain. If that trust is misplaced, too bad.

The consequence of this observation is that there are two highly important controls that must be in place.

**Only use CAs for purposes that make sense**

First, you must be sure about any CA you trust, very sure. Most people will not even know where to find the list of trusted CAs. Here is just an example of what this means. In corporate environments it is custom to use this feature by adding a corporate CA to the pre-installed list of CAs. Anytime you use SSL a corporate server plays man-in-the-middle: it issues on the fly a certificate for the target server with a key it controls, while at the same time setting up an SSL connection to the target. Neither client nor server is any wiser. Such a CA often has the following settings for its rights: all purposes, and not modifiable by the user.

![Certificate purposes](image)

**Trust in a CA implies trust in the operational context**

Second, CAs would better be worth the trust, and work under circumstances that do not jeopardize the trust in them. If any CA has a security incident, this may affect the worldwide system. Recovery from such an incident may mean all (yes, all) systems depending on the CA must take action. The second threat is of a different nature but real nevertheless: a CA may be coerced to participate in setting up a backdoor. Governments can do that. Combine the above two, and you get a very troublesome picture. Any CA in any country where the government believes it has the need and the right to intervene can
do so via the – probably legal in the country- controls over the CA, for servers and clients worldwide. The CA may even be bound to non-disclosure for reasons of national security.

The security measures of the CA must be good enough to merit your trust. A breach of security, by insider, outsider, hack, government intervention or any other means threatens all use of the certificates signed by this CA. Accidents have happened.

Second major flaw: undoing certificate issuing
A certificate is a signed document. Once it exists, there is no telling how many copies float around and where: the cat is out of the bag.
Suppose the impossible happens, and a wrong certificate is issued. Is there any why to correct this situation? At first glance, this is a though problem (it will be at second inspection too, unfortunately).

Expiration, CRL and OCSP
Let certificates expire
A first protection mechanism is that all certificates are time constrained. They are only valid till a well-defined date included in the certificate. If you are lucky this is within a few months, if you are unlucky (like for a CA certificate) it may be 10 years.
It is possible to ignore the expiration date. This may be necessary to access old data, but it is of course a concern that this is possible.
Expirations are also a cause of trouble: there is no early warning system that a certificate is about to expire. It is not uncommon to find people caught by surprise: their certificate is rejected by all other parties and they are the last to know.
In short, expiration dates are more useful to limit key lifetimes than that they serve to limit exposure of a certificate that must no longer be trusted.

Invalidate a certificate – not possible
A certificate is a once given, forever given thing. Yet, keys may get exposed, the user may be declined further use of the keys to enter, etc. Enters the “Oops” options. The first option is to use a “sorry” list: the certificate revocation list (CRL). CAs issue CRLs at regular and managed intervals with certificates that are still valid but which must no longer be trusted. These lists are not small.
Everyone using certificates should (must?) check valid certificates against this list before deciding to accept the certificate. The list must also be the last valid list for that CA, so these lists (1 or more per CA) need to be managed for all CAs by any client.
The second “oops” option is to check on the spot, real time, if a certificate is still worth trusting: OCSP. The CA responds with a signed answer on the certificate status. In principle this must be done for any certificate used in the decision leading to trust: the server certificate, the server-signing certificate, the almost top level certificate and the top level certificate of the CA. On line checks means that the latencies will add up. You are likely not getting any SLAs on response times and availability either.
Wait a second … objective??

Let’s take one step back. We need to have a key exchange solution. We simplify and introduce certificates signed by a limited number (or so it is said) of CAs issuing certificates. Now we go online and inline to the CA to check on the certificate. Would it not be simpler that the server indicates the CA vouching for it, that the server securely connects to that CA and gets the public key, signed by the CA over that secured connection? This is sort of a certificate on the fly: always fresh, always validated. The OCSP response and certificate are replaced by a new assertion, combining both. We are actually replacing the protocol by a new one without the current certificates: they serve no purpose as you have to contact the CA anyway.

Request SSL
Server: www.buzz.com

CA: CAURL

Connect CAURL
Server: www.buzz.com

(www.buzz.com, pubkey: 14AE43...) CASignature

There is no expiration date to manage, no CRL to create and download. An OCSP responder-like component handles it all.

Note that this approach does not eliminate the issue of which CA to trust.