X509box – the manual

August 30, 2016

1 Introduction

X509box is a GUI utility for system administrators who need to generate X.509 certificates, more commonly known as SSL-certificates. These are commonly used to secure websites or e-mail servers against eavedropping by encrypting all connections. The X509box handles the creation of the cryptographic artifacts that a certification authority needs in order to issue a properly signed certifcate.

In order to set up a connection secured by the SSL or TLS protocols, you need a number of items. Almost all of these items can be generated on your own workstation, except for the signed certificate itself. X509box will help you generate the following items:

1. RSA private keys
2. Certificate signing requests
3. PKCS #12 certificate bundles[1]

1.1 RSA private key

The first step of an SSL connection consists of a cryptographic handshake based on the RSA cipher, a so-called public key cipher. The essential feature of this is the fact that a secure connection can be established without the need to transmit secret or confidential information in the clear. This is done through the use of a pair of mathematically related fragments of information called keys.

The private key is a piece of data which the owner of the server should protect as carefully as possible. The actual encryption standard allows you

[1]PKCS #12 will be there in 1.0, it's not present in prereleases yet.

to protect the private key with a password, but that's impractical for servers which run without human interaction from deep inside dark datacenters. You don't want to be present to enter the password every time your server needs to be restarted, like in the deep of night after your operating system finishes installing critical updates.

If you choose not to protect your key with a password, it'll be quite vulnerable when left on your local hard drive. X509box doesn't (yet)[2] provide any means to protect the generated keys in other ways. You'll need to take care of this yourself. Look into encryption solutions that protect your entire hard drive like BitLocker or LUKS. Whatever you do, it's absolutely imperative that your private keys remain just that: private.

Your protection will be completely broken once your private key gets compromised. Do not share it with anyone unless absolutely necessary. Also note that no certification authority ever needs your private key. Immediately stop conducting business with any certification authority that requests your private keys from you. Some certification authorities offer 'easy' web-based services to generate everything for you in one go. Don't ever use such a service, though. It'll give the certification authority and any of its affiliates an easy and undetectable way to eavesdrop on your communications or to impersonate you flawlessly. This cannot be stressed enough: generate your private keys locally and keep them protected at all times.

1.2 Certificate signing request

The certificate signing request (CSR) is a fragment of data containing a bunch of metadata about your planned certificate as well as its public key. The public key is the inextricable counterpart of your private key, but it serves a completely different purpose. The public key is hosted openly on the server you intend to protect, free for all to download and use. Hence the term 'public key', there's nothing secret about it.

The metadata consists of fields for items like country, state, city, organization etc. These will later be used to identify your server to its users. They're intended for human readers.

A certificate signing request is a combination of these metadata fields and your public key. This package is saved into a text file and sent to a certification authority for it to sign, if it decides to trust you. The CSR is closely related to the private key. If you ever lose your private key, any CSR or certificate based on it will instantly become useless. The other way around

[2] X509box may get a secure storage mechanism for keys in a later version. This isn't yet planned though.

is not a problem. Any private key can be used to generate a CSR with a new set of metadata. This is, however, very bad practice. Once your certificate expires, make it a point to generate a completely new set of cryptographic artifacts for its replacement.

1.3 Signed certificate and chain

A signed certificate consist of your certificate signing request, being the bundle of metadata and a public key, cryptographically signed by a certification authority. A certification authority is usually an external company that receives your CSR, verifies your identity to a certain degree and attaches its seal of trust to your request. The certification authority does this by using its own private key to mathematically sign your CSR file. This signature can then be verified by anyone who has the certification authority's public key.

A certification authority is usually built up from a hierarchy of authorities that sign eachother's public keys, forming a chain that cascades down from a single root all the way to your own signed certificate. Those who later connect to your system will download your certificate, figure out who signed it and attempt to verify this signature. To make this easy, your server may offer the certificate authority's public keys right along with its own all the way up to the root.

A collection containing your private key, your signed certificate and any signing authorities above it, can be installed on your server separately or bundled together in PKCS #12 files for use. Linux and UNIX servers tend to use separate files, while Windows seems to have a preference for PKCS #12 files.

1.4 PKCS #12 bundles

PKCS #12 is a cryptographic standard that's used to bundle all manner of cryptographic artifacts together into a single file, much like a ZIP-file. X509box doesn't generate these yet, so you'll need to create them by different means. The OpenSSL toolkit is an option for this but many servers, like the ubiquitous Apache web server, work perfectly well without it.

2 Using X509box

X509box is currently only useful in the preparation stage of generating an X509 certificate. This process consists of a number of discrete steps.

1. Enter your certificate's metadata;

2. Choose a size for your keys;

3. Generate a private key;

4. Generate a certificate signing request (including a public key);

2.1 Certificate metadata

Certificate metadata consists of a number of fields which are filled according to a set of conventions. X509box allows you to set the following common set of fields, which are the most common fields present in certificates, on its first tab[3]. These fields are intended to reflect the certificate holder's identity, be it personal or organizational or a mixture of both.

Country The country contains the ISO language code of the certified entity's country. Most certification authorities only sign if they can verify that the entity's current legal country of residence matches with the information in this field.

State or province The state or province field contains a geographic designation which may obviously contain items like state or province but interpretation is liberal enough for use with counties, districts, dpartements or other geographical groupings between a city and a country.

City or locality This contains your current city or municipality of legal residence.

Organization Contains the formal name of your organization exactly as it's legally registered. Certification authorities check this extremely carefully. The slightest difference in punctuation and capitalization may well be reason not to process your certificate signing request.

Organizational unit The organizational unit is a field for your internal use. You're welcome to fill it with the name of your department, or of the department responsible for the service which uses the certificate.

Hostname or common name For server this is usually the fully qualified domain name as registered in DNS. This field is also often used to store a person's e-mail address for S/MIME e-mail encryption.

[3]The current prerelease version writes dummy information to the CSR if you don't modify each of them at least once, even if it's just to empty them again. This is a bug, since as a feature it's now been documented.

2.2 Key size

The size of your cryptographic keys is measured in bits. The longer the keys in a pair are, the more difficult it will be to crack them through the use of brute force (ie. guessing). The tradeoff is in the computational complexity of actually working with these keys. The longer you make them, the harder CPU's will need to work in order to process and use them. This may be an issue between systems which require lots of short-lived connections to be encrypted between them. Longer keys introduce considerable latency in this scenario in exchange for stronger security.

X509box follows common best practice by not allowing the generation of keys shorter than 2048 bits, which is also the default. The application allows you to choose sizes up to 8192 bits, which should remain secure for at least a number of decades to come. A key this long should only be used to protect secrets which absolutely require more than 20 years of protection[4].

For general use in ecommerce applications, the protection of e-mail and other online communications the current default will hold for years to come.

2.3 Generate private key

X509box's second tab allows you to generate a private key. Although it's not strictly required for a private key, you would do well to adopt the habit of first filling out the metadata fields if you intend to actually generate a CSR based on this key.

The actual generation of a private key is simple, just hit the button and the text field will fill with a lot of what appears to be gibberish. This is actually your key, though, encoded in a format which makes it easy to handle as text. Many applications refer to this format as PEM.

If you need a so-called 'armored' private key, for instance because you need to transmit it across public channels before use, you should enter a strong passphrase into the box labeled 'Passphrase'. Remember what you enter here, because the final key will be AES-128 encrypted using this passphrase as a protection. The key and any related certificate will be useless if you don't remember this passphrase. You'll notice the difference between armored and plain keys in the first line of the text field containing the key data. An armored key will have BEGIN ENCRYPTED PRIVATE KEY in its first line whereas an unarmored key will have BEGIN RSA PRIVATE KEY in

[4]If you need X509box to protect you against three-letter agencies, you have other problems and would be well-advised to obtain a decent formal grounding in the theory and mathematics of cryptography before proceeding! X509box is just a tool. A fool with a tool is still a fool.

this place.

You could copy/paste the key from the text field if you need to enter it into some sort of interface, like a web form or graphical application. If you prefer to store it as a file, just hit the 'save' button and you'll have the key stored as a text file.

Please note that if you choose not to save your key and you exit your application, any CSR you may generate later will be completely useless. The same will go for any certificates you buy based upon such a CSR. So apart from keeping the private key a secret, you also must never lose it. Certificate authorities usually do not offer a refund in case you lose your private key, so losing your private key could be a very expensive mistake to make. Once lost, a private key can –in keeping with its purpose– never be recovered.

2.4 Generate certificate signing request

Generating a proper certificate signing request (CSR) requires that you enter valid metadata on the first tab and generate a private key on the second. Once that's done, you can just hit the button to generate a CSR. The tab works much the same as the private key tab. You generate the CSR, which will then be displayed in the text field above the button. This is so you can copy/paste the text into another running application. Or you can hit the 'save' button to write the CSR to a file on disk.

The CSR is the item you send off to a certification authority for signing. Even though the private key and the CSR may appear very similar at first glance, note the first and last lines of text in each of them. The CSR refers to a signing request while the private key also identifies itself as such. Make absolutely certain that you never send out a private key to the certification authority, or to any third-party for that matter.

After your certification authority processes your CSR, you could keep it for when the certificate expires. Using the same combination of CSR and private key is a quick way to get a similar certificate with a renewed period of validity. This is bad practice, though, because it leaves private keys in the same place for very long periods of time. A certificate does not expire without good reason, so when the time comes to renew it you should do so based on a newly generated private key and its corresponding new CSR.