How are ssl certificates verified?

Asked 16 years, 2 months ago Modified 8 months ago

Viewed 183k times

322

What is the series of steps needed to securely verify a ssl certificate? My (very limited) understanding is that when you visit an https site, the server sends a certificate to the client (the browser) and the browser gets the certificate's issuer information from that certificate, then uses that to contact the issuerer, and somehow compares certificates for validity.

- How exactly is this done?
- What about the process makes it immune to man-in-the-middle attacks?
- What prevents some random person from setting up their own verification service to use in man-in-the-middle attacks, so everything "looks" secure?

security ssl certificate x509

Share

Improve this question

Follow

edited Jul 20, 2023 at 1:19

user207421

311k ● 44 ● 320 ● 488

asked Oct 9, 2008 at 17:16

rcreswick

16.8k ● 15 ● 60 ● 70

wst.space/ssl-part-4-tls-handshake-protocol – Anonymous Platypus Jul 20, 2018 at 12:23

3 found this video very useful in understanding the flow youtube.com/watch?v=T4Df5_cojAs – Krishna Jun 20, 2020 at 12:58

1 how certificate works - steves-internet-guide.com/ssl-certificates-explained – user1089766 Aug 1, 2021 at 14:11

1 Knowing that bit rot will happen sooner or later, I nevertheless also suggest browsing through comparitech.com/blog/information-security/tls-encryption which does a great job of explaining exactly what is going on under the hood without being overly technical. It's thorough enough to provide a good understanding of all the steps involved in this — and seeing why TLS is so robust and secure — without, however, requiring a PhD in cryptography or data communications (or both!). – Gwyneth Llewelyn Nov 13, 2023 at 22:13

6 Answers

Sorted by: Highest score (default)

430

Here is a very simplified explanation:

1. Your web browser downloads the web server's certificate, which contains the public key of the web server. This certificate is signed with the private key of a trusted certificate authority.
2. Your web browser comes installed with the public keys of all of the major certificate authorities. It uses this public key to verify that the web server's

certificate was indeed signed by the trusted certificate authority.

3. The certificate contains the domain name and/or ip address of the web server. Your web browser confirms with the certificate authority that the address listed in the certificate is the one to which it has an open connection.
4. Browser and server calculate a shared symmetric key which is used for the actual data encryption. Since the server identity is verified the client can be sure, that this "key exchange" is done with the right server and not some man in the middle attacker.

Note that the certificate authority (CA) is essential to preventing man-in-the-middle attacks. However, even an unsigned certificate will prevent someone from passively listening in on your encrypted traffic, since they have no way to gain access to your shared symmetric key.

Share Improve this answer

Follow

edited Jul 19, 2023 at 5:09

Steffen Ullrich
123k ● 11 ● 150 ● 190

answered Oct 9, 2008 at 17:28

Eli Courtwright
193k ● 68 ● 218 ● 257

---

4 Around step 1.5 the server also "signs" something with the private key associated with its certificate. This combines with the name/IP check to assure that only the owning site of the certificate presents it. – Darron Feb 5, 2009 at 2:42

94 To see a complete worked example of this process using Firefox connecting to amazon.com, see moserware.com/2009/06/first-few-milliseconds-of-https.html – Jeff Moser Jun 29, 2009 at 16:05

16 I did not know that my browser comes installed with the public keys of all major certificate authorities. Now I know how my SSL certificates are getting verified without risk of MITM :). Thanks! – OneChillDude Jul 25, 2014 at 18:32

5 server needs to request certificate from CAuthority, so it sends request to it. How could CA be sure the server is valid ? – voipp Feb 25, 2018 at 20:30

6 @voipp: Great question! Historically there have been a few approaches, such as "send an email from `webmaster@<domain-being-verified>` or "Place this file on your domain to prove you own it." However, there have indeed been problems with people getting CAs to issue certificates for domains they don't own - famously someone managed to get a shady CA to issue them a certificate for gmail.com! – Eli Courtwright Mar 5, 2018 at 16:12

89

You said that

> the browser gets the certificate's issuer information from that certificate, then uses that to contact the issuerer, and somehow compares certificates for validity.

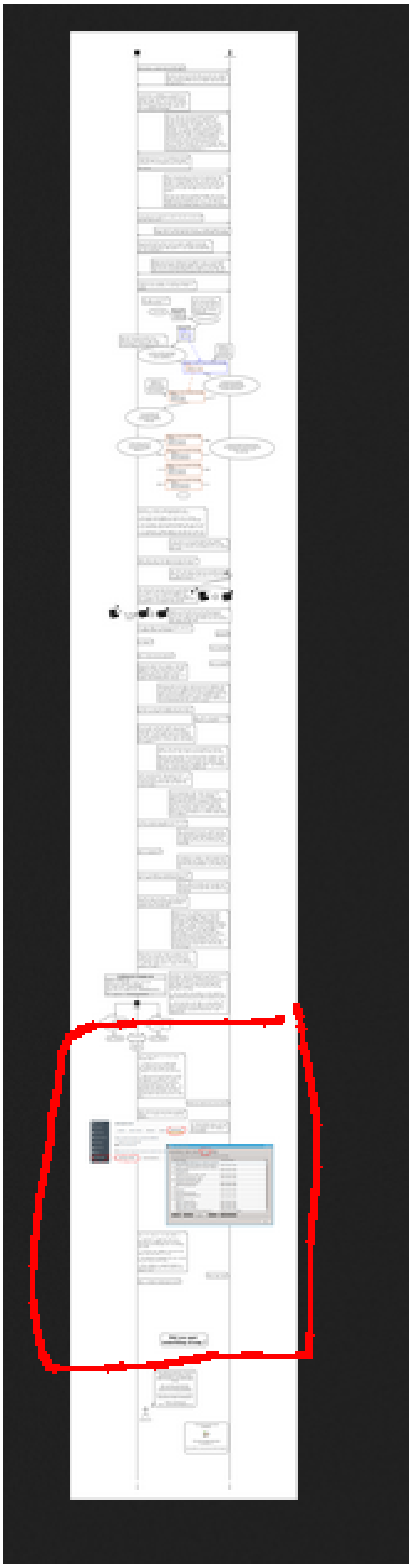
The client doesn't have to check with the issuer because two things :

1. all browsers have a pre-installed list of all major CAs public keys
2. the certificate is signed, and that signature itself is enough proof that the certificate is valid because the client can make sure, on his own, and without contacting the issuer's server, that that certificate is authentic. That's the beauty of asymmetric encryption.

Notice that 2. can't be done without 1.

This is better explained in this big diagram I made some time ago

(skip to "what's a signature ?" at the bottom)



edited Mar 27 at 15:25

Share Improve this answer

Follow

answered May 30, 2016 at 16:06

ychaouche

5,082 ● 3 ● 49 ● 53

17 This should have been the accepted answer. @Eli Courtwright's answer is way to short imho to understand how certificates work. – Felix Crazzolara Jun 1, 2018 at 8:41

4 Fantastic image. Finally something that explains my questions. Everywhere I go to go in depth just said "the browser verifies the certificate is right". BUT HOW DOES IT DO THAT?. This gives an answer. – ori6151 Apr 13, 2020 at 23:31

2 Glorious answer thanks a ton!!!! i dont even care if it leaves out some details, it to my knowledge completely captures all important steps. – Muhammad Umer Oct 20, 2020 at 20:56

1 "All browsers have a pre-installed list of all major CAs public keys": they have a pre-installed list of all major CAs' *certiicates*. The public keys alone are not sufficient. – user207421 Jul 20, 2023 at 1:24

1 StackOverflow seems to have broken the image link (there's only a tiny unreadable blurry image)... I found the original image here and a repo at github.com/ychaouche/SSL-diagram – Ghost4Man Mar 26 at 14:54

73

It's worth noting that in addition to purchasing a certificate (as mentioned above), you can also create your own for free; this is referred to as a "self-signed certificate". The difference between a self-signed certificate and one that's purchased is simple: the purchased one has been signed by a Certificate Authority that your browser already knows about. In other words, your browser can easily validate the authenticity of a purchased certificate.

Unfortunately this has led to a common misconception that self-signed certificates are inherently less secure than those sold by commercial CA's like GoDaddy and Verisign, and that you have to live with browser warnings/exceptions if you use them; **this is incorrect**.

If you securely distribute a self-signed certificate (or CA cert, as bobince suggested) and install it in the browsers that will use your site, it's just as secure as one that's purchased and is not vulnerable to man-in-the-middle attacks and cert forgery. Obviously this means that it's only feasible if only a few people need secure access to your site (e.g., internal apps, personal blogs, etc.).

Share Improve this answer Follow

edited Apr 16, 2020 at 10:43

answered Feb 5, 2009 at 2:16

clint
14.5k ● 13 ● 72 ● 80

3 Indeed, securely distributing your own certificate is one way to skin the cat, but a much easier one is to go to any one of a number of so-called "open" CAs. CACert.org is my favorite. So long as you trust the steps they take to safeguard their cert issuance, then importing their root cert is safe. – nsayer Apr 8, 2009 at 23:36

6 I love this comment - unfortunately it highlights a very important weakness with CAs. Let's say you import a CA cert from Bob Smith - well Bob Smith can sign a certificate for any domain (including google.com and chase.com). This is actually why GoDaddy/Verisign pay big money to be included in the OS - they are vetted by a security outfit to ensure that they have checks in place to make sure they don't sign a cert for a malicious person. I think that you should be able to say "this CA can only sign certs for mysite.com". – Natalie Adams Nov 26, 2013 at 22:15

1 Isn't self signed certificate more secure, since the CAs out there could be paid to sign something they shouldn't have. If you can safely distribute the CA certs to the end-points, always go with self-signed certs. – javaPhobic Mar 17, 2015 at 0:46

Are there any CA that are free and verified in most major browsers? I'm looking for a basic cert merely for verifying I own an email and domain name. The ones I've found aren't in most major browsers though. – Alex Kwitny Sep 11, 2015 at 18:01

@NathanAdams In *theory* the big CAs are supposed to vet requests to keep from issuing bogus certs as you describe... but read this story: stripe.ian.sh – nsayer May 22, 2018 at 15:52

27

I KNOW THE BELOW IS LONG, BUT IT IS DETAILED, YET SIMPLIFIED ENOUGH. READ CAREFULLY AND I GUARANTEE YOU'LL START FINDING THIS TOPIC IS NOT ALL THAT COMPLICATED.

First of all, anyone can create 2 keys. One to encrypt data, and another to decrypt data. The former can be a private key, and the latter a public key, AND VICERZA.

Second of all, in simplest terms, a Certificate Authority (CA) offers the service of creating a certificate for you. How? They use certain values (the CA's issuer name, your server's public key, company name, domain, etc.) and they use their SUPER DUPER ULTRA SECURE SECRET private key and encrypt this data. The result of this encrypted data is a SIGNATURE.

So now the CA gives you back a certificate. The certificate is basically a file containing the values previously mentioned (CA's issuer name, company name, domain, your server's public key, etc.), INCLUDING the signature (i.e. an encrypted version of the latter values).

Now, with all that being said, here is a **REALLY IMPORTANT** part to remember: your device/OS (Windows, Android, etc.) pretty much keeps a list of all major/trusted CA's and their **PUBLIC KEYS** (if you're thinking that these public keys are used to decrypt the signatures inside the certificates, **YOU ARE CORRECT!**).

Ok, if you read the above, this sequential example will be a breeze now:

1. Example-Company asks Example-CA to create for them a certificate.
2. Example-CA uses their super private key to sign this certificate and gives Example-Company the

certificate.

3. Tomorrow, internet-user-Bob uses Chrome/Firefox/etc. to browse to https://example-company.com. Most, if not all, browsers nowadays will expect a certificate back from the server.

4. The browser gets the certificate from example-company.com. The certificate says it's been issued by Example-CA. It just so happens to be that Bob's OS already has Example-CA in its list of trusted CA's, so the browser gets Example-CA's public key. Remember: this is all happening in Bob's computer/mobile/etc., not over the wire.

5. So now the browser decrypts the signature in the certificate. FINALLY, the browser compares the decrypted values with the contents of the certificate itself. **IF THE CONTENTS MATCH, THAT MEANS THE SIGNATURE IS VALID!**

Why? Think about it, only this public key can decrypt the signature in such a way that the contents look like they did before the private key encrypted them.

How about man in the middle attacks?

This is one of the main reasons (if not the main reason) why the above standard was created.

Let's say hacker-Jane intercepts internet-user-Bob's request, and replies with her own certificate. However, hacker-Jane is still careful enough to state in the certificate that the issuer was Example-CA. Lastly, hacker-Jane remembers that she has to include a signature on the certificate. But what key does Jane use to sign (i.e. create an encrypted value of the certificate main contents) the certificate?????

So even if hacker-Jane signed the certificate with her own key, you see what's gonna happen next. The browser is gonna say: "ok, this certificate is issued by Example-CA, let's decrypt the signature with Example-CA's public key". After decryption, the browser notices that the certificate contents don't match at all. Hence, the browser gives a very clear warning to the user, and it says it doesn't trust the connection.

Share Improve this answer Follow

answered Jul 19, 2020 at 9:47

Francisco Vilches
3,728 ● 1 ● 20 ● 19

3 good summary. I still have one question. "Lastly, hacker-Jane remembers that she has to include a signature on the certificate. " => is the signature also not available publicly in the cert sent by the server ? – SRIDHARAN Aug 31, 2020 at 15:41

3 @SRIDHARAN I like your hacker thinking : -) You could copy/paste the signature from the original cert. However, Jane needs to decrypt the web traffic. Only way is Jane puts her own public key in the cert. Then browser uses that key to encrypt requests. Jane uses her private key to decrypt traffic. What happens if Jane copy/pastes signature: 1. Bob's browser uses Example-CA's public key to decrypt signature 2. Browser compares the decrypted signature contents with contents of cert. 3. Browser notices the public keys don't

match 4. Browser tells Bob it's insecure connection.
– Francisco Vilches Sep 1, 2020 at 1:30

Thanks for responding back. I was going through these topics. Now I have a good understanding. I also got confused this with DNS spoofing. For which I found the perfect answer here. security.stackexchange.com/a/94335. – SRIDHARAN Sep 1, 2020 at 6:46

When I learned about HTTPS, I was taught that the server's private key is used to decrypt and the public key is used to encrypt. Is the terminology reversed for certificate verification? The public key refers to the key used to decrypt, and the CA's private key is used to encrypt. Correct?
– Guy4444 Jan 15, 2021 at 18:59

1 You are conflating encryption with digital signing. It's not the same thing. You are also making the same mistake as several others in asserting that the browser has a list of CA public keys. It has a list of CA *certificates*. Their public keys alone are not sufficient. – user207421 Jul 20, 2023 at 1:26

12

The client has a pre-seeded store of SSL certificate authorities' public keys. There must be a chain of trust from the certificate for the server up through intermediate authorities up to one of the so-called "root" certificates in order for the server to be trusted.

You can examine and/or alter the list of trusted authorities. Often you do this to add a certificate for a local authority that you know you trust - like the company you work for or the school you attend or what not.

The pre-seeded list can vary depending on which client you use. The big SSL certificate vendors insure that their root certs are in all the major browsers ($$$).

Monkey-in-the-middle attacks are "impossible" unless the attacker has the private key of a trusted root certificate. Since the corresponding certificates are widely deployed, the exposure of such a private key would have serious implications for the security of eCommerce generally. Because of that, those private keys are very, very closely guarded.

Share Improve this answer Follow

edited Oct 9, 2008 at 17:39

answered Oct 9, 2008 at 17:24

nsayer
17k ● 4 ● 36 ● 52

>> unless the attacker has the private key of a trusted root certificate. I don't think so, Even if attacker possess the private key of non-root certificate (OR) with the help of non-root CA company, then also MITM attack is possible. Google was once vulnerable to MITM attack and the reason was the intermediate CA, may be they signed spoof certificates and gave them to hackers who then intercepted the call and send the spoof certificate and then browser encrypts the symetric key with spoof public key and sent as response. Now, hackers can decyprt this symetric key using his own privatekey – Manikandan Kbk DIP Jul 31, 2022 at 18:08

@ManikandanKbkDIP you are correct, but it’s much easier to recover from a compromised intermediate CA - the root (or a CA above) can revoke the cert and nothing else need be done. If a root is compromised, then every client in the world must update their trust store. – nsayer Aug 1, 2022 at 19:10

The client has a pre-installed store of CA *certificates.* Their public keys alone are not sufficient. – user207421 Jul 20, 2023 at 1:27

@user207421 I disagree. The public key is sufficient to verify the signature of the child certificate. And for *root* certificates, it is *unlikely* (though not impossible) that the rest of the meta-data in the self-signed root certificate is required to establish trust in the children. *Most* implementations store the certificates, but this isn't strictly necessary. – nsayer Jul 21, 2023 at 2:11

10

if you're more technically minded, this site is probably what you want: http://www.zytrax.com/tech/survival/ssl.html

warning: the rabbit hole goes deep :).

Share Improve this answer Follow

answered Jun 22, 2016 at 19:27

Mike Frysinger
3,043 ● 1 ● 23 ● 28