Lecture 15

TLS and Secure Channels

March 15, 2021

1 Common Crytographic Network Protocols

1.1 TLS (Transport Layer Security)

TLS (Transport Layer Security), originally called the Secure Socket Layer (SSL), is used to provide an encryption wrapper around HTTP to make HTTPS. TLS is wrapped around the application layer so it is often utilized when securing many other application layer protocols.

The main security goals of TLS are to: authenticate the server, ensure the confidentiality and integrity of the traffic, and to ensure that the client is infact connected to the server which they think they are connected to.

1.2 SSH (Secure Shell)

SSH is a modern day alternative to the antequated Telnet. Telnet was phased out by the more secure SSH due to security reasons as Telnet uses an unencrypted connection that sends information in plaintext, which is easily exploitable.

In terms of the security benefits that SSH provides, SSH authenticates both server and client, and ensures the confidentiality and integrity of the traffic, between server and client.

1.3 IPsec (Internet Protocol Security)

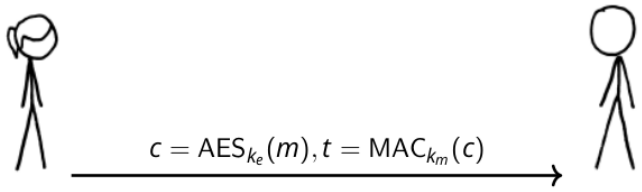
IPsec is an encrpyted and authenticated alternative to IP. It is a complicated set of protocols which attempt to replace the IP layer. Regular IP is an insecure solution because the packets which travel that layer are unencrypted. IPSec is a common solution in VPN (Virtual Private Network) services.

IPSec's security goals are client and server authentication, the authentication of headers and option to encrypt headers, and ensuring the confidentiality and integrity of payloads.

2 Constructing a Secure Encrypted Channel

In order to construct a securely encrypted channel, several preliminary steps must be performed:

2.1 Encryption and MAC (Message Authentication Code)



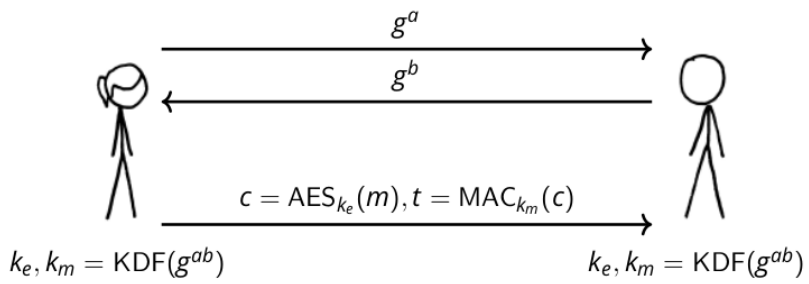
Alice (left) and Bob (right) want to communicate on a secure channel that is protected against passive easedroppers and man-in-the-middle attacks.

Assuming Alice and Bob have shared a set of keys, Alice sends her AES ciphertext and the MAC of the ciphertext to Bob. Bob can now check the MAC and decrypt the cipher text to get the original message.

The MAC is a short string of data which authenticates the message, i.e. it confirms that the message was sent by the true sender and was not modified in transit.

2.2 Diffie-Hellman key exchange

In order to negotiate charing encryption and MAC keys, there must be a Diffie-Hellman key exchange.



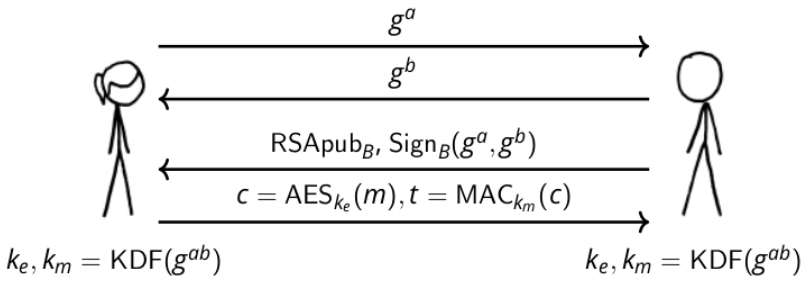
Note: In the image above g^a should be read as $g^a \ mod \ p$ and likewise g^b should be read $g^b \ mod \ p$

If a Diffie-Hellman key exchange has occurred then both Alice and Bob will have a **shared secret**. In this case Alice and Bob's shared secret is g^{ab}.

Using this shared secret Bob and Alice can use a Key Derivation Function (KDF), $k_e, k_m = KDF(g^{ab})$, which we can think of as a hash function that is used to create encryption and MAC keys they can use for symmetric cryptography.

2.3 Ensure Authenticity of Endpoints

There is a vulnerabiliy with this approach because if there is an active man-in-the-middle attack, the attacker can intercept the Diffie-Hellman key exchange. To ensure the authenticity of the endpoint we must use **digital signatures**, to prevent man-in-the-middle interference of the key exchange. You can have either one or both parties sign the key exchange with a long-term public key.

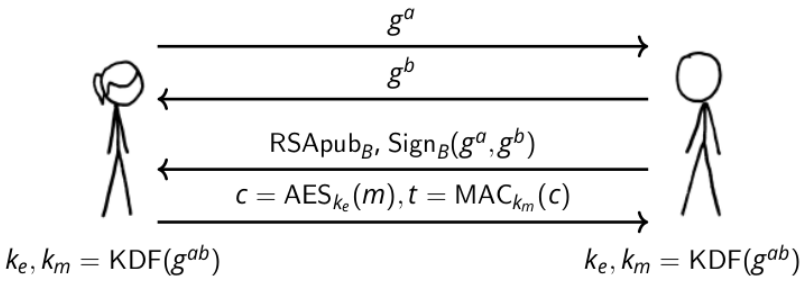


For example, assume that Alice is the client (web browser), Bob is the web server, Alice knows the server's long-term key, and Alice is trying to communicate with Bob.

To protect the Diffie-Hellman key exchange Bob will sign the key exchange with his long-term key. Since Alice knows both Bob's original long term-key and the signature which Bob has given Alice, she can verify the signature by using Bob's public key before progressing with the encryption.

2.4 Trusting Signatures

While we may now be protected against man-in-the-middle attacks targeted towards the Diffie-Hellman key exchange, we have still not verified the integrity of Bob's public signing key which Alice has received.



The public signing key that Bob sends to Alice is susceptible to active man-in-the-middle attacks, so we must determine a way to trust the keys that the client receives.

Assuming that there is a man-in-the-middle who is intercepting all of the communications between Alice and Bob, an attacker can substitute Bob's public key with their own generated public key and Alice would not be able to tell the difference.

Due to the nature of this attack, we must have an **external** method of establishing trust in keys.

2.5 Establishing Trust in Keys

Meet in person to exchange keys.

This is the simplest way, however this is not practical at scale over the internet

Fingerprint Verification

Fingerprint verification is a methodology that follows the TOFO (Trust On First Use) paradigm of verifying the cryptographic hash of a public key through a seperate channel. It is sometimes used to secure SSH host keys. A benefit of this method is that you can trust the fingerprint forever as long as you dont believe that the connection was man-in-the-middle attack during the first connection.

Hard Code Public Keys in Software

Otherwise known as "Certificate pinning" this method is used by web browsers to protect against attacks from malicious sites. Pinned certificates are usually keys baked into software. To figure out which certificates to trust there are 'certificate authorities' whom act as a commercial trusted intermediary that verfiy public keys and sign them in exchange for money. If you trust the certificate authority then you must also trust the keys they sign. You do not need to worry about these certificate authorities decrypting your traffic because all they know you are doing is signing the public key, they have no idea of the private key used.

Web of Trust

If you do not like the premise of trusting large commerical companies with profit in mind you can instead choose to create a Web of Trust. This is done by trusting your friends, and in turn trusting anyone who has had their public key signed by one of your friends so you can indirectly

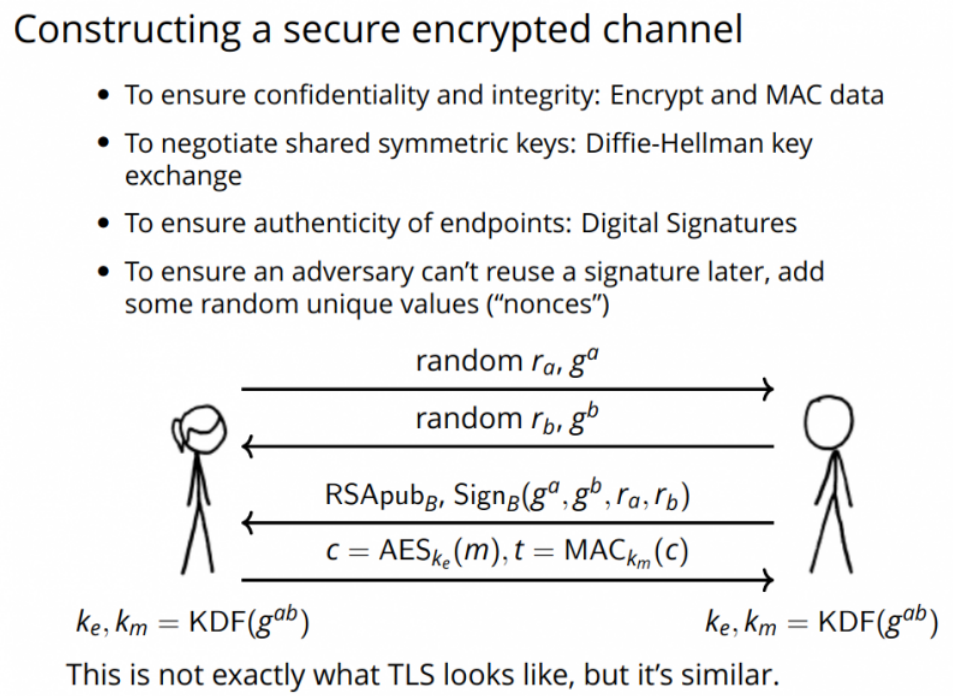
trust others through your friends. This method of establishing trust is used by PGP (Pretty Good Privacy), an open source email encryption software written in the early 1990s.

```
nadiah$ gpg --edit-key rivest@csail.mit.edu
gpg> trust
pub  1024D/567B4BAD  created: 2010-12-19  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  1024g/EFE31B86  created: 2010-12-19  expires: never       usage: E
[ unknown] (1). Ronald L Rivest <rivest@csail.mit.edu>

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu

Your decision?
```

2.6 Using RNG in Signature

Returning our discussion to the secure channel: assume Alice knows to trust a key because it has a certificate authority signature or it is baked into our software. Now Alice and Bob are able to negotiate a fresh shared session key and encrypt all of their messages using symmetric crypto which will be secure against an active man in the middle and a passive eavesdropper.

One final detail: in order to protect against an adversary from replaying a signature (reusing it across multiple handshakes), you might include some extra randomness, for example a nonce, a value that you use once. It doesn't necessarily have to be cryptographically strong, but it has to be unpredictable to the adversary. The signature is over these randomly chosen values so should be different every time the protocol runs, so adversaries can't steal the signature from a previous protocol.



Keep in mind that the Diffie-Hellman key exchange is subject to a man in the middle attack. Therefore you need to use digital signatures, because you need to know where to get your trust from the public key that is signed, so you need a public key infrastructure. Once you've done that, then you can trust that Diffie-Hellman key exchange hasn't been man in the middle. Then you derive symmetric session keys and then use symmetric crypto to encrypt the content.

3 TLS: Transport Layer Security

3.1 TLS Overview

TLS was called SSL in the 1990s. It provides an encrypted channel for application data. It's used for HTTPS (HTTP inside of a TLS session). Keep in mind that the content that's being sent, like the symmetrically encrypted messages, once negotiation is done, is like the HTTP connection. There are several protocol versions which are not numbered sequentially. Academic papers show that maintaining support for old versions of the protocol lead to bad downgrade and decryption attacks.

SSL 1.0 Terribly insecure; never released.

SSL 2.0 Released in 1995; terribly insecure,deprecated in 2011.

SSL 3.0 Released in 1996; terribly insecure,deprecated in 2015.

TLS 1.0 Released in 1999; deprecated in 2020.

TLS 1.1 Released in 2006; deprecated in 2020.

TLS 1.2 Released in 2008. Ok.

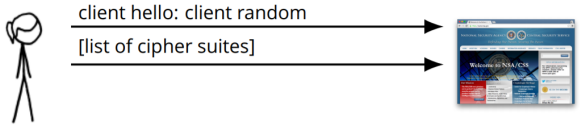
TLS 1.3 Standardized in August 2018 and is being rolled out now; major change from TLS 1.2 .

TLS 1.1, 1.2, and 1.3 are similar, but in general TLS 1.3 is a better protocol. Older versions of TLS have potential flaws.

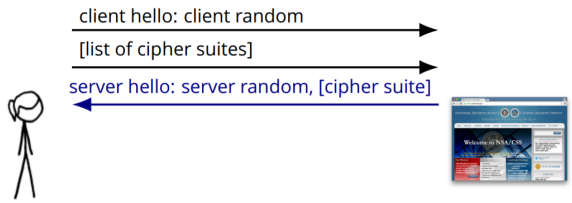
3.2 TLS 1.2 with Diffie-Hellman Key Exchange

3.2.1 Step 1-4

Step 1: The client (browser) tells the server what kind of cryptography it supports.

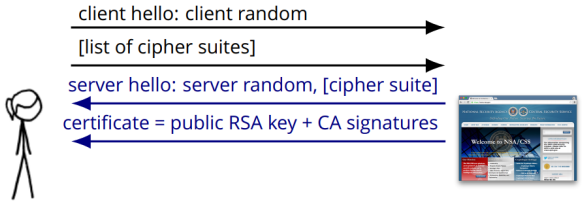


Step 2: The server tells the client which kind of cryptography it wishes to use.



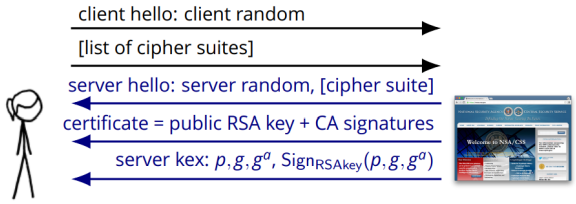
TLS 1.2 with Diffie-Hellman Key Exchange

Step 3: The server sends over its certificate which contains the server's public key and signatures from a certificate authority.



TLS 1.2 with Diffie-Hellman Key Exchange

Step 4: The server initiates a Diffie-Hellman key exchange.

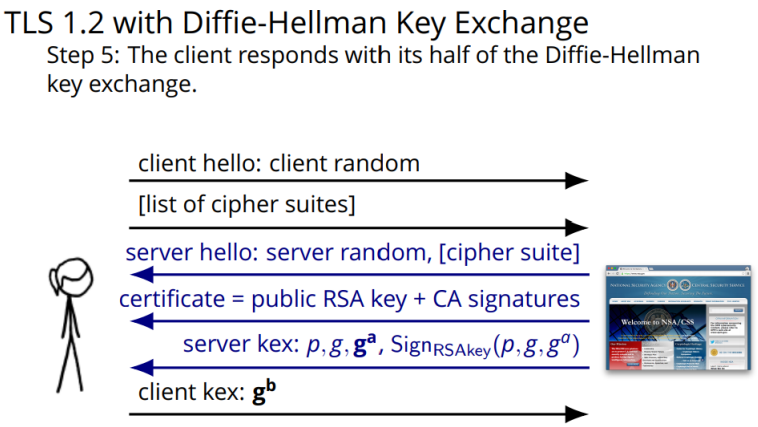


To protect against man-in-the-middle attacks, the server uses its public key to sign the Diffie-Hellman key exchange.

TLS also allows client authentication, but this is rare.

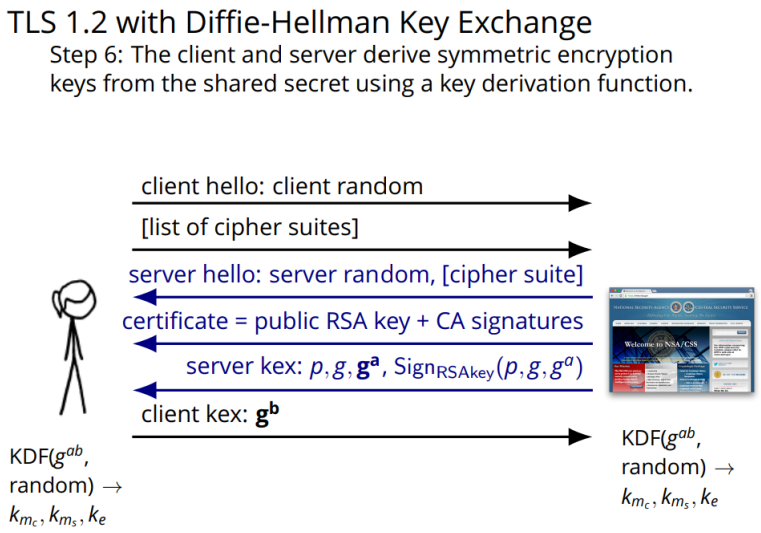
The server will sign its part of the key exchange, shown as $Sign_{RSAkey}(p, g, g^a)$ in the image above. Since the message is signed by the server's RSA key, which is signed by a certificate authority that Alice trusts, it cannot be modified by a man-in-the-middle attack.

3.2.2 Step 5



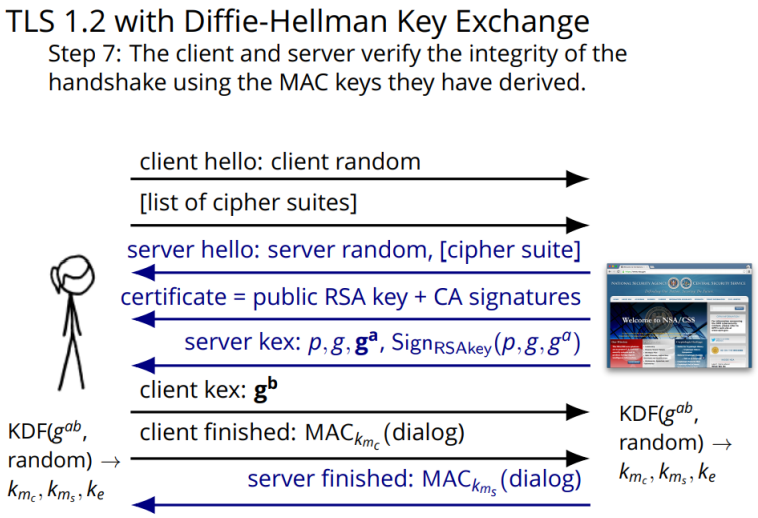
This message is usually not authenticated since user authentication is not commonly done at the cryptographic layer when using the web.

3.2.3 Step 6



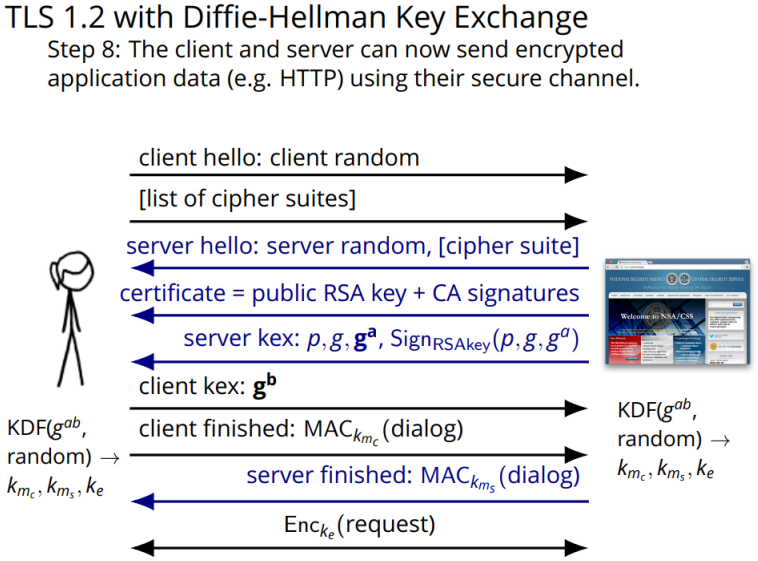
KDF is the key derivation function, k_{m_c}, k_{m_s}, and k_e are the symmetric keys, k_{m_c} is the client's MAC key, k_{m_s} is the server's MAC key and k_e is the encryption key.

3.2.4 Step 7



They can verify the integrity of the handshake by each sending a MAC (Message Authentication Code) of all of the messages they have sent so far using the symmetric keys that they derived in the last step. This guarantees that the channel is secure.

3.2.5 Step 8



For example, Alice (client) will make an HTTP GET request for index.html over the encrypted channel.

3.3 Certificates and Certificate Authorities in TLS

Sample Certificate

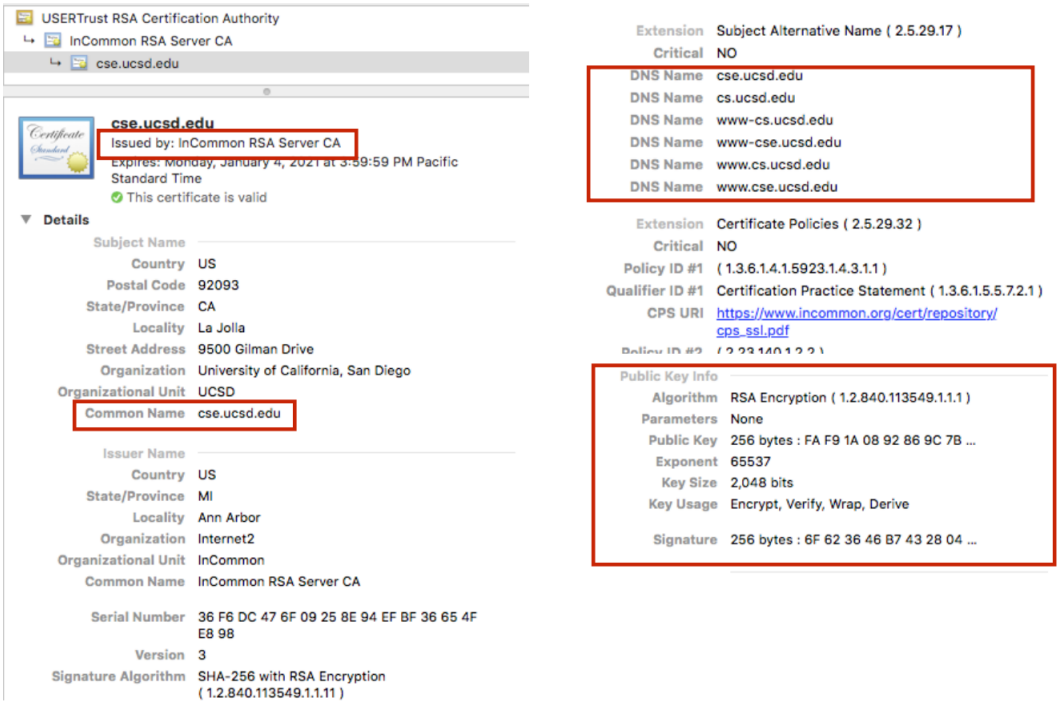


Figure 1: an example of a valid certificate from cse.ucsd.edu

The certificate authority (CA) is InCommon RSA Server CA.

The domain name for which this certificate is meant is cse.ucsd.edu, with multiple DNS names listed

The public key uses RSA Encryption, with exponent 65537, N = 2048, and the public key and signature are both view able to the user.

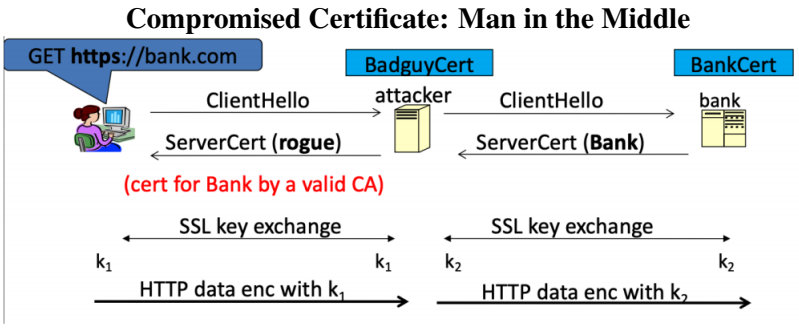


Figure 3: Man-in-the-middle attack using rogue certificate

Figure 2: This part of the certificate tells the user where to look for revocation information

3.4 Revocation

It's possible for a key to be compromised, ie for an attacker to obtain a private key. This is clearly undesirable, as any private/encrypted messages sent to whomever owns this private key can now be read by the attacker, and/or the attacker can sign any messages he wants as if he were the original owner of the private key. In the case that a Certificate Authority key is compromised, a client may receive a perfectly valid certificate that was signed by an attacker, and there would be no way of knowing this certificate is invalid unless the CA announced that their public key is no longer valid.

Therefore, there exists a revocation process for public keys. If Alice, for example, finds out that her private key has been compromised, she is able to announce or allow her senders to find out that her public key is no longer valid and they should not be using that public key to encrypt messages to her anymore. This verification must be signed by the private key, as you wouldn't want anyone else to arbitrarily cancel your key. Both CA and PGP PKIs support this.

The attacker has managed to obtain valid certificates from a CA trusted by the client.

This allows the attacker to intercept the data being sent from the bank, and send the client his own ”certified trusted” public key as if he were the bank; the client will accept this, (since it has the valid certificate) encrypt her data using this malicious key, and send the data to the bank.

This gives the attacker free reign to proxy any and all traffic to and from the client and the bank, modifying any data as he pleases, as he can decrypt whatever the client was sending to the bank.

One could imagine the attacker modifying the client request to say something along the lines of ”transfer all the money to the attacker account.”

3.5 CA Hacks / Vulnerabilities

There is a long history of CAs getting hacked or certifying wrong things.

2011: Comodo + DigiNotar hacked to issue fraudulent certificates for Hotmail, Gmail, Skype, Yahoo Mail, Firefox

2013: TurkTrust issued fraudulent certificate for Gmail

2014: Indian NIC issue certs for Google and Yahoo!

2016: WoSign issues cert for GitHub.

Mitigations

Certificate pinning: Limits risk by restricting which certificates are considered valid for a particular website. Instead of allowing any CA in the trusted list to sign a certificate, the browser pins the CA of choice; any certificates received that are not from this pinned CA are considered invalid. Therefore, attackers are unable to man in the middle even if they have a valid certificate from a trusted authority as long as the pinned authority doesn't match.

Certificate Transparency: All certificates are publicly disclosed, providing greater insight and transparency. Certificate transparency has two main components, logs and monitors.

Logs maintain records of all issued certificates to a domain. These logs are append only, and cannot be altered or deleted once a certificate has been added to the log (using something called a Merkle Tree to achieve this)

Monitors do just that, monitor the logs for anything they can be set to monitor. Some monitors can allow users to create and run queries for certificates, as google does. Some domains may be interested in receiving notifications for when a certificate is issued to their domain, or if it matches some query they are interested in monitoring.

3.6 TLS 1.2 with RSA Key Exchange

Another way to negotiate keys in TLS 1.2 is using RSA public key encryption to share a secret master key.

1) This starts with the Alice[client] sending a hello.

2) Then, the server will send over its certificate, which contains its public key.

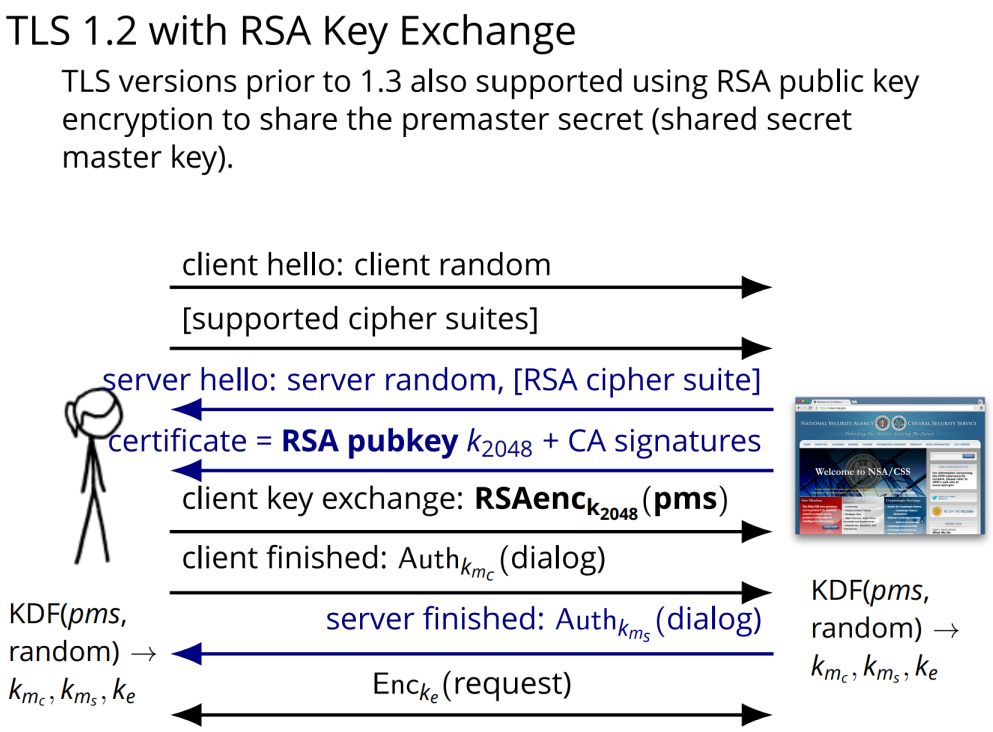
3) Alice[client] will choose a random value, called a premaster secret, which she encrypts to the server's RSA public key and sends back to the server.

4) The server is the only one who can decrypt the value of the encrypted premaster secret.

5) The server and client will then use a key derivation function to derive their encryption and MAC keys.

6) The server and client will exchange MACs of the dialogue.

7) Finally, they will send encrypted requests using the symmetric encryption key.



The only difference from using the Diffie-Hellman key exchange is the client instead encrypts a secret and sends it to the server.

3.7 How TLS Achieves Its Security Goals

It allows transmitting information between a client and the server with an encrypted connection. Listening in, only allows the recipients who are exchanging information to be seen. Without the secret keys the information cannot be decrypt it.

3.8 What If a Private Key Gets Stolen or Compromised?

The attacker is able to actively man-in-the-middle a connection. Therefore, they are able to impersonate the server to others. This allows the attacker to eavesdrop between the parties. With the a valid secret key they are able to decrypt messages that are being sent from the users compromised key.

With a compromised RSA key, the attacker can obtain the session keys and decrypt the sessions information. Any past traffic that is recorded can also be decrypted with such an attack.

3.9 TLS v. 1.2 and Below Vulnerabilities

TLS 1.2 and below allowed attackers manipulate data between client and servers. Including credit card information, credentials, and other valuable information.

TLS 1.2 addressed these concerns. They allowed clients and servers to choose specific hash and signature algorithms. Allowed authenticated ecryption for other data modes.

3.10 TLS 1.3

Removes the support of SHA-224 and MD5 because of the compromised vulnerabilities.

Removes RSA key exchange, which allowed protection for passive decryption

Allowing handshake encryption immediately after key exchange. This limits the amount of data that an eavesdropper can see. Also a noticeable slower time to secure a connection.

3.11 TLS Key Theft and Other Risks in the Wild

A very common attack is SSL tripping. The attacker gets in between the user and an encrypted host. They act as a middle host between the client and secured website. Any information the user enters will first go through the attacker

Using expired TLS certificates can cause a man-in-middle attack. Therefore, any expired certificates should be removed from servers.

Not identifying any forges or untrusted certificates

3.12 The “Crypto Wars” and the Historical Development of TLS

Unofficial term used to describe the feud between governments and their attempt to limit the access cryptography strong enough to go unrecognized by intelligence agencies. In 1975 the U.S. introduced the Data Encryption Standard, but led many to believe there will always be a back door if a government wanted to eavesdrop on ”encypted” data.

The development of TLS has been supported by governments in order to keep secure connections between clients and servers, while keeping with government standards.

3.13 TLS Keytheft and Other Risks in the Wild

If an attacker was able to issue validated certificates for a domain they don't own, then they wouldn't need to get a hold of any existing private keys. They would just make their own key pair and issue a certificates for it. The attacker DoS's the victim's server and stands their own server up using the attacker's certificate and key.

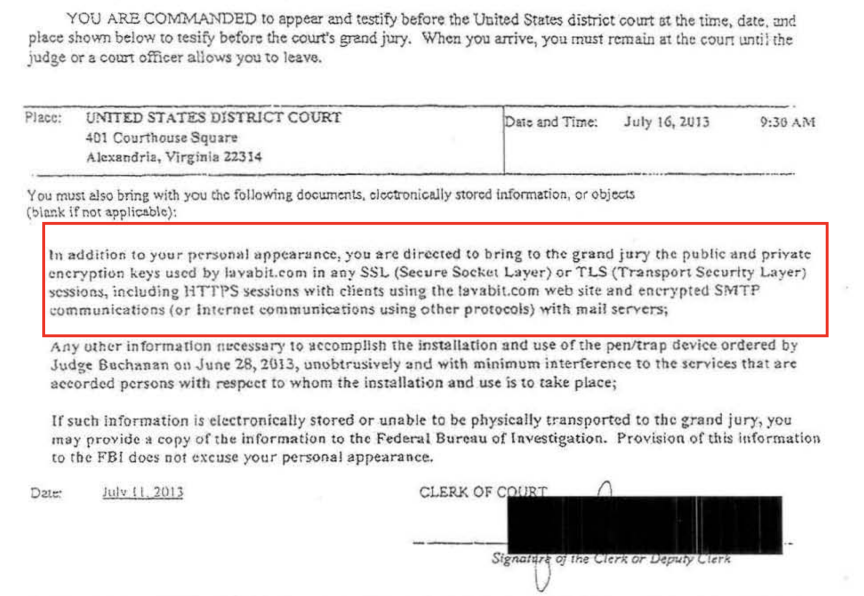
However, if an attacker gets the private key for a victim's server then the attacker does not need to create a new certificate. The attacker will just grab the existing cert (it's already being handed out by the victim's server) and DoS the victim's server and stand up their own server using the victim's cert and key.

Often an ordinary user is vulnerable to one or more parties that they don't really trust, between them and any particular peer out on the Internet, this might be the proprietor of the coffee shop where they're using WiFi, or their home ISP, the network engineers at their employer's place of business, or even the sovereign government of the country they're in.

3.14 Lavabit

Even if TLS and key exchange works perfectly, there is still the matter of old-fashioned extortion by powerful governments. For example, consider the events of 2013 after Edward Snowden blew the whistle on the NSA's domestic spying capabilities.

Snowden was using a secure mail service Lavabit. In order for the US government to spy on his communications after his defection. Lavabit was commanded to appear in court, and to bring all of their TLS, HTTPS, and SSL keys to present to the court. However, Ladar Levison chose to bring a blurry, illegible photo which he claimed to be the TLS key.



YOU ARE COMMANDED to appear and testify before the United States district court at the time, date, and place shown below to tesify before the court's grand jury. When you arrive, you must remain at the court until the judge or a court officer allows you to leave.

Place:	UNITED STATES DISTRICT COURT 401 Courthouse Square Alexandria, Virginia 22314	Date and Time:	July 16, 2013	9:30 AM

You must also bring with you the following documents, electronically stored information, or objects (blank if not applicable):

In addition to your personal appearance, you are directed to bring to the grand jury the public and private encryption keys used by lavabit.com in any SSL (Secure Socket Layer) or TLS (Transport Security Layer) sessions, including HTTPS sessions with clients using the lavabit.com web site and encrypted SMTP communications (or Internet communications using other protocols) with mail servers;

Any other information necessary to accomplish the installation and use of the pen/trap device ordered by Judge Buchanan on June 28, 2013, unobtrusively and with minimum interference to the services that are accorded persons with respect to whom the installation and use is to take place;

If such information is electronically stored or unable to be physically transported to the grand jury, you may provide a copy of the information to the Federal Bureau of Investigation. Provision of this information to the FBI does not excuse your personal appearance.

Date: July 11, 2013

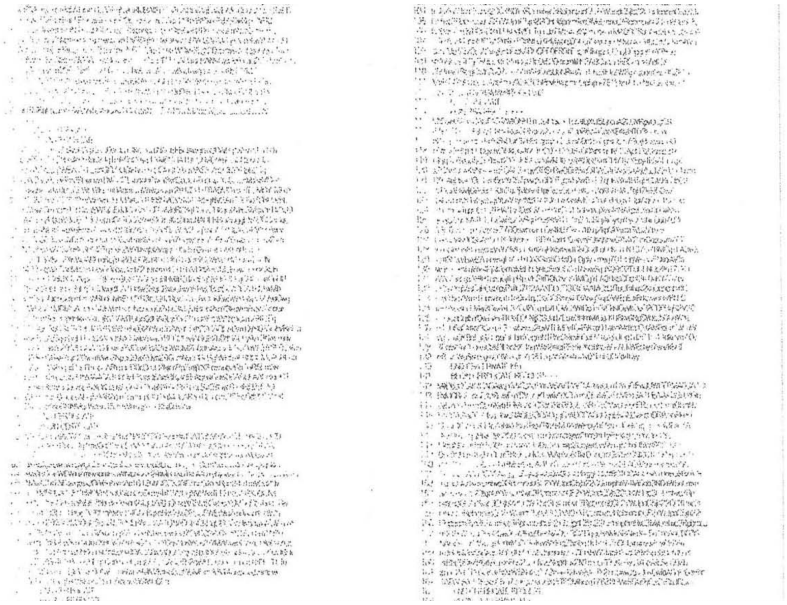
CLERK OF COURT

[redacted]

Signature of the Clerk or Deputy Clerk

Lavabit summons

Lavabit never gave up the key, and instead chose to end their business operations rather than violate the privacy of their customers around the globe. Politicians and the US government do not care about the individual's right to privacy. For example, here is a quote from former president Barack Obama:



The image of the key brought into court

"Because, if, in fact, you can't crack that [encryption] at all, government can't get in, then everybody is walking around with a Swiss bank account in their pocket – right? So there has to be some concession to the need to be able to get into that information somehow."
It is with this spirit that the US government has actually weakened encryption to make their job of spying easier. For example, in the 1990s, TLS 1.0 included options to weaken the protocol for US export control (browsers outside of the US were to request the weakened option). Even though these options are no longer required due to political shifts, there are still servers that respect the request. This lead to a series of attacks such as FREAK, logjam, and DROWN.