Encryption and SSL Certificates Guide

Steve Hayes

Version 1.02, February 2015

This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of the license, visit http://creativecommons.org/licenses/by/3.0/deed.en_GB.

In essence, you can modify, distribute, copy, print and use the guide as you wish for commercial or non-commercial purposes so long as you include an attribution to the original author. Please also include a reference to drenewydd.com.

For further information, please contact steve@drenewydd.com.

TRADEMARKS/REGISTERED TRADEMARKS
Any computer hardware and software brand names mentioned in this guide are protected by their respective trademarks and are acknowledged here.

Table of Contents

1 Introduction....2
1.1 Conventions....2
1.2 Revisions....2
2 Symmetric encryption....3
2.1 The players....3
2.2 A childish cypher....3
2.3 An unbreakable cypher....5
2.4 A new player....5
2.5 Block cyphers....6
2.6 Brute force attacks....7
2.7 Why encrypt?....8
2.8 Key escrow....8
3 Public Key encryption....10
3.1 The RSA cypher....10
3.2 Digital signatures....11
3.3 Man in the Middle....11
3.4 Practical encryption....12
3.5 Practical signing....13
3.6 Detached signatures....13
3.7 Public encryption key length....14
3.8 Blocking the Man in the Middle....14
3.9 Other public key algorithms....15
3.10 Ephemeral keys....15
3.11 Steganography and Traffic Analysis....16
4 SSL and certificates....18
4.1 Certificate Authorities....19
4.2 Other attacks....19
5 Passwords....21
5.1 Three tricks that make Mal's job harder....22
6 Making keys and certificates....23
6.1 Make an RSA private key....23
6.2 See the key contents....24
6.3 Get the RSA public key and hash....24
6.4 Encrypt and decrypt a short message....25
6.5 Make a self-signed certificate....25
6.6 Encrypt and decrypt a file....26
6.7 Install a key and certificate in Apache....26
6.8 Get a RapidSSL certificate....27
6.9 Install a key and certificate in Dovecot/Postfix....29

1 Introduction

You may need to encrypt sensitive information or set up a secure website. You may only be trying to make sense of scary stories you've seen in the media (e.g. "***In a major security breach, millions of passwords have been stolen from Acme Corporation.***").

In either case, you're likely to encounter a lot of jargon and misinformation and a lot of statements and recommendations that don't seem to fit together and make sense.

This guide aims to make it all understandable and to demonstrate that there's a lot more to security than just the number of bits used in encryption.

Chapter 6 includes exercises where you can use open-source tools to make public encryption key pairs and make and install website certificates.

1.1 Conventions

When an important term is introduced, it's shown in **bold**.

> *There are paragraphs formatted in italics like this one. They contain information that may be interesting, useful and helps you to understand the steps being described.*

1.2 Revisions

Compared to version 1.01, this version has the following changes:

- This section (1.2) is added.
- Section 3.5 now mentions including the originator's own key.
- Section 3.10 now includes Diffie-Hellman key exchange.
- Section 3.11 is added.
- Section 5.1 now includes the use of global salt.
- In Chapter 6, the example key size is increased to 2048 bits.
- Section 6.9 is added.

2 Symmetric encryption

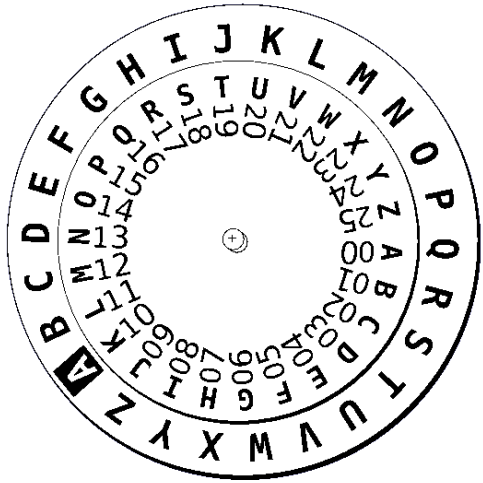
2.1 The players

Alice (person A) wants to send a message to **Bob** (person B). Alice and Bob know that an eavesdropper (**Eve**) can see or hear the message as it's being sent. They'll use a cypher to thwart her.

> *If Alice and Bob use prearranged words to hide the meaning of the message from Eve, e.g. Alice tells Bob that Aunt Ethel has a new budgie, this is a **code**.*
>
> *If Alice uses a process that turns the message into gibberish that only Bob should be able to make sense of, this is a **cypher**. This document covers cyphers.*

When Alice turns the message (**plaintext**) into gibberish (**cyphertext**), she **encrypts** it. When Bob turns the cyphertext back into the plaintext message, he **decrypts** it.

2.2 A childish cypher



This is a simple encryption device that's used to entertain a child but it

illustrates some important concepts. It consists of two disks made of card that can pivot against one another. Each disk has the letters of the alphabet written around it.

Ignore the numbers 00 to 25 for a moment.

As shown here, the letter A (white on black) on the larger disk lines up with K on the smaller disk. You can encrypt a word by finding each letter on the larger disk and replacing it with the letter alongside on the smaller disk. For example, with the disks set as shown, the word SECRET is encrypted as COMBOD. Lining up the same letters and reversing the process retrieves the original word.

What the disks are doing is Modulo 26 addition. Encryption can also be done by replacing each letter with the corresponding number on the smaller disk, adding the key, subtracting 26 if the result is over 25, then turning the resulting number back to its corresponding letter.

*As shown with A and K lined up, the **key** is 10 (K).*

The letter E corresponds to the number 4. Adding 10 gives 14 which corresponds to O.

The letter S corresponds to the number 18. Adding 10 gives 28. That's over 25. Subtracting 26 gives 2 and the corresponding letter is C.

A computer can easily do this sort of modulo arithmetic.

The encryption is easily broken. There are only 26 keys (ways the disks can be set) and one of them does nothing as each letter lines up with itself. It isn't even necessary to try all the other 25 possibilities. The letter E is the most common in English and lining up the most common letter in the encrypted message (O in our example) with E on the larger disk usually reveals the key.

There are ways to improve things. The letters on the smaller disk could be put in a different order and the disk could be turned after each letter is encrypted. The encrypted message could be encrypted again using another different pair of disks. Extend these ideas a bit further, build them into a machine that makes things easier and you have the famous Enigma that Germany used in WW2.

Of course, Enigma wasn't as secure as the Germans thought...

2.3 An unbreakable cypher

Each letter in the message can be encrypted using a different randomly-chosen letter as the key. These key letter sequences are printed on a **one-time pad** with both Alice and Bob having a copy.

> *For example, the word SECRET encrypted with the one-time pad KLAHEW is CPCYIP.*

So long as certain requirements are met, the scheme is provably **unbreakable**. Eve can use any computer she likes, even quantum ones not yet invented. She'll never know what the messages say and here's why:

When Eve sees an encrypted message, she imagines a vast number of possible plaintext messages it could represent.

> *The message should have some extra characters added (padding) so Eve can't know its exact length.*

For each of these plaintext messages, there's a key letter sequence that would turn it into the cyphertext that she saw. The key letters are chosen completely randomly and all the sequences are equally possible. There is no way for her to tell which plaintext message is the right one.

> *To make the cypher unbreakable, Alice and Bob must use a truly random method of making the one-time pads, they must make sure Eve can never get a copy of them or of the unencrypted messages and they must never ever use the same pad more than once.*
>
> *Because of the difficulty of making the pads and sending them securely, one-time pad encryption is not used much. Governments and secret agents have used it (and doubtless sometimes still do) for their most sensitive communications.*

2.4 A new player

Alice sends Bob another message: MALOWESYOUTWENTY-POUNDS. She's encrypted it with a one-time pad so Eve can't know what it says.

This time it isn't Eve she needs to worry about, it's the malefactor **Mal**. He can guess what the message says and he's going to make some

changes...

Suppose the first letter of the encrypted message is R. Mal turns this into the corresponding number 17. He subtracts the number for M (12) and adds the number for Y (24). The result is 29 so he subtracts 26 and changes the first letter of the encrypted message to D.

Mal changes some more letters, then sends the encrypted message on to Bob.

Bob knows that the one-time pad system is unbreakable so when he sees the message YOUOWEMALSEVENTYPOUNDS, he pays Mal without hesitation. Foolish Bob!

> *Since Mal knows the plaintext message, he can calculate the one-time pad sequence. He can change the message to anything he likes and, if Alice ever makes the mistake of reusing the one-time pad, he'll be able to decrypt that message.*

2.5 Block cyphers

Up to now, we've been encrypting plaintext a letter at a time but most modern encryption is done on blocks of letters (or other data) using a **block cypher** such as **DES (Data Encryption Standard)** or **AES (Advanced Encryption Standard)**.

For example, Alice can use 128 bit AES to encrypt a 128 bit block of data (perhaps representing 16 successive letters in a message) with a 128 bit key. Bob knows the key and he can use AES in decryption mode to recover the data.

If Mal changes even a single bit in the encrypted (cyphertext) message, Bob will only see gibberish when he decrypts it. He'll know that the message has been tampered with.

However, this doesn't stop Mal from **replaying** whole blocks of cypher-text. If Alice tells Bob to PAYMALTENPOUNDS, Mal can send Bob a dozen copies. Maybe Bob will fall for that one too.

There are ways to guard against this type of attack. Blocks can be chained together so changing the plaintext in any one of them changes the cyphertext for all the ones after. Including a date and time or a sequence number at the start of the message lets Bob see if a message has been repeated.

2.6 Brute force attacks

When Eve sees a message that Alice has encrypted with a block cypher, she can try each possible key in turn until she finds one that decrypts the message to something that makes sense.

That's almost certainly the right key. Not only can she decrypt this message but now she can decrypt any others that Alice and Bob send using the same key.

Eve's problem is that there are so many different keys. There are over 340000000000000000000000000000000000000 possible 128 bit keys. If she sets a million computers to work and each one tries a million keys each second, it'll take over a billion billion years to try half of the possible keys and have a 50% chance of finding the right one.

Even if Eve gets hold of computers that are billions of times faster, all Alice and Bob need to do is to use a longer key. Just one more bit doubles the number of possible keys. Doubling the key length squares the number of possibilities (pay attention, Dan Brown!).

> *Of course, Eve would never try to decrypt the messages by brute force. Instead, perhaps she could hack Alice or Bob's computer to get the key or the messages. She could hide outside with binoculars and try to see the messages while they are being typed or read. Or she could invite Alice or Bob out and get them drunk. A chain is only as strong as its weakest link...*
>
> *A secure block cypher is one where there is no way of finding the key or decrypting a message that's faster than* ***brute force****.*
>
> *It's long been suspected that DES has weaknesses that are known to the American NSA but, despite many years of analysis by independent cryptographic experts all over the world, none have ever been reported other than the problem of the 56 bit keys being too short. AES might have weaknesses that allow GCHQ and NSA to snoop despite its longer keys but, again, the experts haven't reported any.*
>
> *For the purposes of this document, it isn't necessary to go into block cyphers in more depth.*

2.7 Why encrypt?

Once upon a time a well-known writer of airport novels was up in court charged with insider trading. His wife was a director of a company that had been taken over. A friend bought shares in the company just before the takeover bid was announced, then sold them on at a big profit before he'd even paid for them.

The novelist's wife testified that she'd never spoken to him about the bid but it turned out that documents relating to it had been faxed to her at home.

The documents would have been hanging out of the fax machine for who knows how long before she collected them but no-one could prove that the novelist had read them and he was acquitted.

> *More recently he spent time as a guest of Her Majesty after being convicted of perjury in an unrelated case.*

Everyone knew that the novelist had had the opportunity to see the documents but still he couldn't be convicted. Just think of all the people in telecommunications and in the intelligence agencies who can see your messages without you ever knowing. Do you really think that an agency which has trafficked drugs and arms to fund its dubious activities would hesitate to swap insider information for money or other favours?

If you're sending sensitive information, you need to use strong encryption.

2.8 Key escrow

Eve is working for the government. She's worried that she can't actually decrypt many messages. Her life would be so much easier if people were forced to give copies of their keys to the government.

Of course this would cause an uproar. She'll try to avert this by proposing a system of **key escrow** where she has to show a legitimate need before she gets hold of a person's key.

Perhaps Eve isn't all that clever but there are smarter people around her who can see a snag with this.

What happens when someone encrypts their message with a truly secret key, then encrypts the resulting message with their escrowed key?

Eve wants to read the message. She gets the escrowed key and decrypts it but all she gets is another encrypted message. She doesn't have that key.

The sender is breaking the law and could be hit with a fine if the police and courts can be bothered. But doing that tips off the sender and recipient that they are under surveillance. And Eve will be in much bigger trouble herself if she can't publicly justify her decision to get the escrowed key.

No wonder key escrow quietly died a death many years ago.

3 Public Key encryption

In Chapter 2, we saw methods where the same key is used to encrypt and to decrypt messages. These are known as **symmetrical key** cyphers. They work well where only two people are involved.

Let's introduce Carol. Now there are three pairs of people who might need to communicate: Alice and Bob, Alice and Carol and Bob and Carol.

Introduce Dot as well and there are 6 pairs. Introduce Ed and there are 10 pairs. With 10 people, there are 45 pairs.

Everyone could share the same key but that means that they can all read each other's messages. Even if everyone is entirely trustworthy, Eve has lots of chances to steal the key.

Alternatively, there could be a different key for each pair of people. The safest way for a pair to agree on a key is to meet face to face. It's all sounding cumbersome.

Suppose that there's a cypher that uses different keys to encrypt and decrypt and that, even if Eve knows the encryption key, there's no way faster than brute force for her to decrypt a message encrypted with it.

Now, each person needs just one secret (private) decryption key unique to them. They can give the matching encryption key to anyone who wants it, even Eve and Mal. It's a **public key**.

Anyone wanting to send Alice a message encrypts it with her public key and sends it to her. Only Alice has the **private key** that'll decrypt it.

3.1 The RSA cypher

RSA, devised by Ron Rivest, Adi Shamir and Leonard Adleman, was the first secure public key cypher. It's still the best known.

An RSA public key consists of a **modulus** (*m*) which is the product of two large randomly chosen prime numbers, along with a more or less arbitrarily chosen **public exponent** (*e*), usually quite small (e.g. 3, 35 or 65537).

The corresponding private key consists of the same modulus (*m*) along

with a large **private exponent** (*d*). This is calculated from *e* and the two prime numbers: it cannot be calculated from *e* and *m* without factoring *m*.

The security relies on the fact that, while it's not hard to multiply the two prime numbers together to get *m*, there's no way other than trial and error to factor *m* and recover the primes. Trial and error is calculated to take far too long (e.g. billions of years).

Choosing the primes and calculating *d* involves a lot of advanced mathematics which I don't pretend to understand. However, the encryption and decryption steps themselves are not complicated.

For any value *P* that's less than *m*, $(P^e)^d = P$ when the arithmetic is performed modulo *m* (Section 2.2). The information that's to be encrypted is represented as the number *P* and this is raised to the power *e* (modulo *m*) to obtain the cyphertext. Decryption is achieved by raising the cyphertext to the power *d* (modulo *m*), recovering *P*.

3.2 Digital signatures

RSA has another extremely useful feature. As noted above, $(P^e)^d = P$. But $(P^e)^d = P^{e \times d}$ and $e \times d = d \times e$ so $(P^d)^e = P^{d \times e} = P^{e \times d} = (P^e)^d = P$.

This means that a message that's encrypted with the private key can be decrypted with the corresponding public key. Anyone who successfully decrypts a message using Alice's public key knows that it must have been encrypted with Alice's private key. Only Alice has access to that so the message must be from her.

This looks like the answer to Mal's tampering but...

3.3 Man in the Middle

Bob receives a message, apparently from Alice. It seems to have been encrypted with her private key but he doesn't have Alice's public key. He asks Alice for it.

Alice sends it to him but Mal intercepts it. He replaces Alice's public key with his own before sending it on to Bob. When Bob uses it to decrypt the message, it reads SENDMALONEMILLIONPOUNDS. Will Bob fall for this one?

Mal can also decrypt and read any messages that Bob tries to send to Alice. He has the private key corresponding to what Bob thinks is Alice's public key. He can re-encrypt the message with Alice's real public key and send it on to her so she won't suspect anything.

These are **man-in-the-middle** attacks. Alice and Bob can guard against them by meeting face to face to exchange keys but that rather defeats the object of public key encryption.

> *Later on, we'll see other ways to protect against man-in-the-middle attacks.*

3.4 Practical encryption

Public key encryption requires a lot of calculations, especially when the private key is being used. To speed things up, the message itself isn't usually encrypted this way.

Instead, the encryption program randomly chooses a **session key**, then encrypts the message with it using a symmetric algorithm such as AES (Section 2.5). The session key is encrypted using the public key algorithm, then it's appended to the encrypted message and the whole thing is transmitted.

Decryption involves finding and decrypting the session key using the public key algorithm, then using it to decrypt the message itself.

This is much quicker because only the session key itself needs to be encrypted and decrypted using the slow public key algorithm and there's another advantage too.

If the same message is being sent to several people, the session key is appended multiple times, each time encrypted with a different recipient's public key. The encrypted message itself is the same for each recipient.

> *A copy of the session key encrypted with the public key of the person sending the message is often included so that they can decrypt the message they've sent.*

3.5 Practical signing

To speed things up and make them more flexible, messages aren't normally signed by encrypting them with the signer's private key.

Instead, a mathematical algorithm is used to generate a hash of the message. The hash itself is encrypted with the signer's private key and appended to the message.

The signature is verified by decrypting the appended hash using the signer's public key, then using the same hash algorithm as before to calculate the value for the message itself. So long as the two values agree, it's assumed that the message hasn't been tampered with and the signature appears to be valid.

> *A simple **hash algorithm** might convert each character in the message to a number, then add all the numbers together. If Mal changes any single character in the message, the sum will change and the tampering will be spotted. However, he can swap characters in the message or make additional changes so the sum stays the same.*
>
> *Instead, much more sophisticated algorithms such as **SHA (Secure Hash Algorithm)** are used. These have a **one-way** characteristic which effectively prevents anyone from making two different messages with the same hash value.*

3.6 Detached signatures

An interesting characteristic of hash-based signatures is that they can be separated from the message itself.

For example, suppose that Alice and Bob make a secret agreement. They need to notarise it but they don't want anyone (even the notary) to know what it says.

This isn't a problem at all. They generate and exchange **detached signatures** of the agreement. Each of them keeps a copy of the agreement and can verify that the other has signed it correctly. They can go together with the detached signatures and have them notarised without the notary seeing the agreement itself. Only if the agreement breaks down need it be produced in court, along with the notarised signatures. The court can verify the signatures and see when they were notarised.

3.7 Public encryption key length

In Section 2.6, we saw that a 128 bit symmetric key is unbreakable in practice. A 256 bit one can probably never be broken even by a computer the size of the universe.

Public key systems such as RSA normally use longer key lengths, e.g. a modulus of 1024 bits or more. This is because not all modulus values are possible and because it's not entirely clear if there might be ways to speed up brute force attacks.

> *Some years back, a major problem came to light. Because of a bug in the random number routines, Debian-based systems (including Ubuntu) were only generating 65536 possible different key pairs. It wasn't hard to generate them all, put them in a file, then look for a public key in the file and grab the corresponding private key.*
>
> *The problem wasn't immediately obvious and, once it was spotted, a lot of work was needed to fix it. Not only did all Debian systems need patching to fix the bug but all their keys and certificates had to be replaced with newly generated ones.*

3.8 Blocking the Man in the Middle

In Section 3.3, Mal replaced Alice's key with his own so that he could send messages to Bob that appear to come from Alice and could decrypt any messages that Bob encrypted with what he thought was Alice's key.

Bob can guard against this attack by generating a hash of Alice's key, then calling her on the phone to confirm the hash (assuming Mal can't intercept the call and successfully imitate Alice's voice).

The well known **PGP (Pretty Good Privacy)** public key system uses this method along with a **web of trust**. For example, suppose Alice receives Carol's public key and verifies it with her face to face or over the phone. She signs Carol's key and sends it to various people including Bob. Bob already has Alice's public key and has verified it. He trusts Alice so, when he gets Carol's key, sees that Alice has signed it and verifies the signature, he can use it without having to verify it with Carol herself.

Alice doesn't even have to send the signed public key to Bob. She can send it back to Carol and Carol can put it on a website. Anyone can download it and, if they have a trusted copy of Alice's key and trust Alice herself, they can use Carol's key with confidence.

A key can include many signatures: the recipient only needs to trust one of them (or maybe partly trust two of them).

*There's an open-source version of PGP. It's called **GPG (GNU Privacy Guard)**.*

3.9 Other public key algorithms

In the USA, RSA was covered by a patent (now expired). Elsewhere at that time (and even now in many countries), algorithms couldn't be patented. Programs using RSA were written that couldn't be used legally in the USA.

This, along with delight in pure mathematics and desire for fame and glory led to other one-way mathematical problems being used in competing public key algorithms. For example, GPG uses El Gamal encryption which is based on a discrete logarithm problem.

3.10 Ephemeral keys

There are ways to send a message where Eve can capture all the communications, use the full force of the law to demand the keys and still can't decode the message.

One approach would be for Alice to ask Bob to generate an **ephemeral** (temporary) public/private key pair. He does this, signs the ephemeral public key with his usual private key and sends it to Alice.

Alice verifies the signature, then encrypts a message with the ephemeral public key and sends it to Bob. She destroys the message.

Bob decrypts the message with the ephemeral private key, reads it, then destroys both it and the ephemeral private key.

Eve captured and recorded the ephemeral public key and the encrypted message but neither she, Alice, Bob or anyone else can decrypt the message since the private key needed to do this doesn't exist any more.

*This encryption technique offers what is known as **Perfect Forward Security**. The simpler **Diffie Hellman** method, commonly used in **Internet Key Exchange**, does the same. It's a bit like RSA but older and works as follows:*

Alice and Bob agree a value and modulus. These don't have to be secret although keeping them so prevents a man-in-the-middle attack.

Alice randomly chooses an exponent (a), raises the value to it and sends the result to Bob. Bob also chooses an exponent (b), raises the same value to it and sends the result to Alice.

Alice and Bob each raise the result from the other to the exponent they themselves had chosen. Because

$$(V^a)^b = (V^b)^a$$

both get the same result. They use this as a key for their subsequent communication. Eve doesn't know either of the exponents that they chose so she can't calculate the key. Once the two exponents and the key have been destroyed, no-one can decrypt the messages.

3.11 Steganography and Traffic Analysis

When Eve sees that Alice and Bob are exchanging messages, she may suspect that something is afoot. If the messages are encrypted, she is going to be even more suspicious.

Alice and Bob can try to avoid Eve's suspicions by hiding their secret messages in things that appear innocuous. This is known as **steganography**.

> *As well as being a code, the message about Aunt Ethel's budgie in Section 2.1 might be an example of steganography.*

There are various ways to hide information in files. For example, the least significant bit of the brightness of each pixel in a photograph can be replaced with a bit of an encrypted message. This causes some image graininess but Eve might think that the camera wasn't very good.

Tracking who is communicating with who without seeing their actual messages is known as **traffic analysis**. For example, this can identify

networks of suspected terrorists.

> *Eve might dabble in a little insider dealing if she spots that top executives of two apparently unrelated companies are communicating (Section 2.7).*

Since the Snowden revelations, companies such as Google, Facebook and Twitter have strengthened encryption of connections with their users (Chapter 4) and may have reduced or eliminated intelligence agency access to their servers. Intelligence agencies' concern that terrorists, etc. can communicate undetected through these servers is not without foundation.

> *Recently, the British Prime Minister made a fool of himself by seemingly calling for encryption to be banned. If this actually happened, the collapse of the banking system as just one example would soon follow. However we can't expect a politician to actually understand what he is talking about. Maybe he was just concerned about the loss of access to these communications. The real answer is for the agencies to accept that they will have to do without wholesale trawling of people's cat photos and instead reach agreement with these companies to mutually track real threats.*
>
> *When the terrorists are being really devious, they'll use steganography to hide messages in the cat photos. If lots of people look at them, the agencies won't know which of them are reading the messages and which ones just like cats. The services can try to prevent this by mangling the pictures just enough to mess up the messages.*

4 SSL and certificates

Alice logs on to www.bankofbob.com to do some on line banking. She types in her user name and password.

Mal would dearly love to know the password so he could transfer all Alice's money to himself. He can't though and here's why.

When Alice's web browser connects to Bank of Bob, the bank sends it a **certificate**. The certificate contains three vital parts:

- The bank's public key.
- Details of the bank, especially its website address (www.bankofbob.com).
- A digital signature of the previous two items taken together. The private key used to make the signature is guarded by an organisation known as a **Certificate Authority**.

Alice's web browser knows the public keys for a number of certificate authorities and, if Bank of Bob has any sense, it has chosen to use one of these. The web browser checks the signature so it knows that Mal hasn't tampered with the certificate since the authority signed it.

The web browser confirms that the website address in the certificate matches the one that Alice typed in.

Provided all this checks out correctly, the browser shows a padlock symbol so that Alice knows that she is safe. The browser generates a new random session key, encrypts it with the public key from the certificate and sends it to the bank. When Alice types in her password, the browser symmetrically encrypts it with the session key before sending it.

Bank of Bob uses its private key to decrypt the session key and uses that to decrypt the password. It and Alice's browser use the session key to encrypt everything else during the session. Mal can't even see how much money Alice has.

> *This system is known as **Secure Sockets Layer (SSL)**. Starting a web address with https:// ensures that SSL is used.*

4.1 Certificate Authorities

SSL security depends on the Certificate Authority. Its primary responsibility is to make sure that it only ever issues a www.bankofbob.com certificate to Bank of Bob itself.

Certificate Authorities use various methods to try to ensure this. The simplest method is completely automatic – they send an e-mail to one of a limited number of addresses at the domain, e.g. postmaster@bankofbob.com. The e-mail contains a password that the applicant uses to complete the application and receive the certificate.

That isn't a terribly secure scheme (though it's the one I used to get the drenewydd.com certificate from RapidSSL). Other authorities require masses of documentation and may issue Extended Validation certificates that show Alice even more reassuring messages, e.g. showing the address in green.

One problem is that there are dozens and dozens of certificate authorities based in countries around the world. Mal can shop around for one that he can pressure or bribe into giving him a bankofbob.com certificate. Then all he needs to do is to tamper with the DNS system, a router or Alice's computer so Alice's connection is redirected to his fake Bank of Bob website.

4.2 Other attacks

Mal could also try one of the following tricks:

- Redirect www.bankofbob.com to the fake website without bothering with a certificate and hope Alice doesn't notice that the padlock symbol is missing.

  Alice's browser may warn her when she starts typing her password into the insecure site.

- Redirect www.bankofbob.com, install a self-signed certificate on the fake site (Sections 6.5 and 6.7) and hope Alice doesn't understand and ignores the warnings that her browser displays.

- Generate his own private key and surreptitiously install the corresponding root certificate on Alice's computer. This allows him to mount man-in-the-middle attacks for any website he likes.

Some companies do this on their own PCs: then they can use a web proxy to monitor everything their employees do. Certain governments could require this too.

- Register bankofbib.com, get a certificate for it and wait for Alice (or someone else) to mistype the address.

  He can help this along by sending Alice a phishing e-mail containing a link to www.bankofbib.com.

- Install malware on Alice's computer. A keylogger would send him every key that Alice presses before her web browser even sees it. He can easily spot her user name and password when she logs in to the real Bank of Bob site.

  Many banks require you to click things on the screen while logging in. That isn't so easy to capture.

- Hack into Bank of Bob and steal the private key or the list of passwords (Chapter 5).

5 Passwords

Mal has decided not to mess with SSL certificates. Instead, he'll try to guess Alice's password at Bank of Bob.

We assume he already knows Alice's user name. If she's chosen an obvious password such as her cat's name or she's used the same password on a less secure site that's already been broken into, Mal will be accessing her account in no time. However, when he makes a wrong guess, the bank makes him wait a while before he can try again. Too many wrong guesses and Bank of Bob will lock Alice's account completely.

But Mal has a friend at the bank who's managed to steal the password file. Surely that'll reveal Alice's password along with those of all the other customers?

Decades ago it might have but the passwords are no longer stored in the file. Instead, they are hashed with a one-way algorithm (Section 3.5) and the hash results are stored. When Alice types her password, the bank's system hashes what she types and compares the result with the stored hash. If they match, the password is accepted.

> *One-way means that, while it's easy to get the hash value from a password, there's no way other than trial and error to get the password from the hash value.*
>
> *Hashing is a much better method than encrypting the passwords in the file. For that to work, the decryption key would have to be stored on the bank's system and Mal's friend might be able to steal it too. If you see references to a file of encrypted passwords, it really means hashed ones.*

The password file isn't useless though. Mal can use a program such as *John the Ripper* that automatically tries millions of guesses just as fast as his computer can calculate the hashes. It'll start with dictionary words and names before moving on to random sequences of letters, numbers and symbols. It may also use a precomputed *rainbow table* of password hashes: these can be found on-line.

The stolen file has the password hashes for every customer: each guess can be compared to all of them. Mal probably isn't specifically targeting

Alice: any customer with an easy to guess password and money in their account will do.

> *If Mal is trying to break into a company system, e.g. to steal trade secrets, he'll be glad to access any account there to glean information he can use in a **social engineering** attack.*

5.1 Three tricks that make Mal's job harder

- Unix/Linux systems used to store the password hashes along with the user names in the */etc/passwd* file. This file has to be readable by many users and programs on the system so they can see the user names. That makes it all too easy to steal. Nowadays the password hash values are stored in the */etc/shadow* file which is only readable by root.
- Some systems don't just hash the passwords once. The hash result is hashed again and that in turn is hashed again. This may be repeated 1000 times. Normal password checks are slowed down but that usually isn't a problem. Mal's job is harder because it takes much longer to try each password guess.
- A common trick is salting the passwords. When a user sets their password, the system chooses additional **salt** characters that it adds to the password before calculating its hash. When the user enters the password, the system finds the salt characters and again adds them before calculating and checking the hash value.

  Often the salt characters are chosen randomly and stored in the same file as the password hashes. If Mal has the hash values, he'll have the salt characters too but they still make his job harder. He can't use a rainbow table: adding entries for each possible word and salt value would make it far too big. Also each user has different salt: he can't hash each password guess once, then compare it to all the hashes in the file.

  An even better approach is to add the user name to the password along with **global salt**. This is the same for all users and can be securely stored separately from the password hashes while the different user names mean users must be attacked individually.

6 Making keys and certificates

To follow this chapter, you should be able to use the Linux command line as described in the LAMP Server Setup Guide. Commands that you need to type in are shown `in a different font like this`.

Sometimes you'll need to type in a specific phrase or look for a phrase on the screen or on a web page. These are shown in italics, e.g. “enter your domain as the *Common Name*”.

> *If you're using something other than Linux, (e.g. Windows), you might be able to find corresponding utilities. A simpler way is to restart your computer temporarily with a bootable Linux disk (these days most are, e.g. Ubuntu), carry out the steps below, then copy the private and public keys and certificates to a thumbdrive.*

6.1 Make an RSA private key

Type the following command:

```
openssl genrsa -out mykey.key 2048
```

This makes a new 2048 bit private key and saves it as *mykey.key*. You can specify a different file name and/or key length.

> *Anyone who manages to steal mykey.key will be able to read your messages, sign as you, etc. You can symmetrically encrypt the key, e.g. by using* `openssl genrsa -des3 -out mykey.key 2048` *instead. You'll be prompted for a passphrase twice to guard against typing errors. The encryption key used will be a hash of the phrase you type.*
>
> *If you choose to encrypt the key, you'll need to type in the passphrase whenever it's used. That's always a bit inconvenient and is definitely a bad idea if it's to be used with a server certificate or other automatic use where you may not be around when needed.*
>
> *-des3 specifies **triple DES** encryption. This uses two different 56 bit DES keys, avoiding the problem of the DES key being too*

short. The plaintest is encrypted with the first key. The result is decrypted with the second key and the result of that is encrypted again with the first key. The alternating encryption and decryption steps mean that, if the two keys are the same, the result is identical to that with standard DES.

6.2 See the key contents

Type the following command:

```
openssl rsa -text -noout -in my.key
```

The two primes, modulus and public and private exponents appear on the screen.

E.g. a 2048 bit modulus is shown as 257 bytes (8 bits each). Each byte is shown as two hexadecimal digits. The first byte is the most significant one and is always zero. The last byte is the least significant and is always odd (primes larger than two are always odd and products of odd numbers are always odd too).

6.3 Get the RSA public key and hash

Type the following command:

```
openssl rsa -in my.key -pubout >mykey.pub
```

The public key is saved as *mykey.pub*: change the file names as needed.

If you chose to encrypt the private key, you'll be asked to type in the passphrase.

If you leave out >mykey.pub, the public key appears on the screen instead of being saved as a file.

Type the following command:

```
openssl dgst -md5 mykey.pub
```

The **MD5** hash of the public key is calculated and shown on screen.

The MD5 algorithm is older and slightly less secure than SHA. You can use `-sha` *instead of* `-md5`.

If you send the key to someone else, they can use the same command to see the hash of the key file they receive. They can call you to compare the hashes and confirm that the file hasn't been tampered with (Section 3.8, man-in-the-middle).

6.4 Encrypt and decrypt a short message

This section uses RSA directly (without symmetric encryption and a session key) to encrypt and decrypt a message that's shorter than the key modulus. It's for interest only - you can skip it if you want.

Type the following command:

```
openssl rsautl -out message.enc -pubin -inkey
mykey.pub -encrypt
```

Type your message, then press CTRL+D. The message is encrypted and saved as *message.enc*. Type the following command:

```
openssl rsautl -in message.enc -inkey my.key
-decrypt
```

The message is decrypted using the private key and shown on the screen. If you encrypted the key, you'll be asked to type in the passphrase.

If you use openssl rsautl -out signed.enc -inkey my.key -sign, the message is encrypted with your private key and saved as signed.enc. Anyone can read it using openssl rsautl -in signed.enc -pubin -inkey mykey.pub -verify – they know that only you could have sent it.

6.5 Make a self-signed certificate

Type the following command:

```
openssl req -new -x509 -key my.key -out my.crt -days
3660
```

This makes an **X509** certificate valid for 3660 days (10 years) and saves it as *my.crt*. You'll be asked for various bits of information. Most of them can be left blank (by entering .) but be sure to enter the domain you'll use the certificate for (e.g. www.example.com) as the *Common Name*.

If you haven't made an RSA key (Section 6.1) or you want a different key for the certificate, use openssl req -new -x509 -nodes -out my.crt -keyout server.key -days 3660. This makes a new private key, saves it as server.key, then makes the certificate too. -nodes indicates that server.key is to be unencrypted.

6.6 Encrypt and decrypt a file

Choose an existing medium-sized file (e.g. tens or hundreds of kbytes) on your system. Type the following command:

```
openssl smime –encrypt –binary –in testfile –out
xxx.enc my.crt
```

Replace *testfile* with the name and extension of the file you've chosen. A copy of the file encrypted with the public key from *my.crt* is saved as *xxx.enc*. Type the following command:

```
openssl smime –decrypt –in xxx.enc –inkey my.key
–out xxx
```

xxx.enc is decrypted using the private key *my.key* and saved as *xxx*. Type the following command:

```
diff testfile xxx
```

Again replace *testfile* with the name and extension of the file you chose originally. `diff` shows differences between it and *xxx*: there shouldn't be any.

You can use these commands to encrypt a file before you attach it to an e-mail. Modern e-mail client programs usually have built-in encryption and signing. You should find out how to use those instead – it's much more convenient.

6.7 Install a key and certificate in Apache

These instructions assume you've set up your web server as described in the *Lamp Server Setup Guide* and are familiar with how to use `scp`, log in to the server and use Midnight Commander.

Use `scp` to copy the private key file and certificate to the server.

Log into the server and move the two files into */etc/ssl/private*.

Use e.g. `chown root:root /etc/ssl/private/my.key` and `chmod 600 /etc/ssl/private/my.key` to make sure that the private key file is only readable by root.

Go to */etc/apache2/sites-available* in Midnight Commander (having started it with `sudo mc`). Highlight *default-ssl* and press F4 to edit it. Look for the line saying *SSLCertificateFile* and change the file name and path to point to the new certificate file (e.g. */etc/ssl/certs/private/my.crt*). Look for the line nearby saying *SSLCertificateKeyFile* and change it to point to the private key file (e.g. */etc/ssl/certs/private/my.key*). Press F2 to save your changes and F10 to exit the editor.

Type the command

```
/etc/init.d/apache2 reload
```

Go to the https version of your website (e.g. https://example.com) using your usual web browser. You should see a warning that the certificate is invalid. This is because it hasn't been signed by a Certificate Authority (i.e. it's self-signed).

> *You can ignore the warnings and continue to the site. If you're using Firefox, you can choose to suppress the warning when you visit the same site in future: this can be done with Internet Explorer too but it isn't so easy.*
>
> *As mentioned in Section 4.2, you'll need to have your certificate signed by a Certificate Authority to prevent your visitors seeing these warnings.*

6.8 Get a RapidSSL certificate

> *As mentioned in Section 4.1, RapidSSL uses a simple automatic method of verifying that you own your website. This allows their certificates to be issued quickly at a modest price.*
>
> *I'm not necessarily recommending RapidSSL. You may want to use another Certificate Authority but their procedures will be somewhat different.*

Use your web browser to go to www.trustico.co.uk or www.trustico.com. Click on *View RapidSSL Products*. Choose the type

of certificate you want and click *Order Now*.

You need to choose various options and fill in some forms. Eventually, you'll see a box where you need to paste a Certificate Signing Request (CSR). Use the following commands on your computer:

```
openssl req -new -key my.key -out my.csr
```

Follow the recommendations shown on the website when filling in the various fields. Be sure to get the *Common Name* right (e.g. www.example.com). You can leave the *Extra Attributes* fields blank.

> *You can make a new private key at the same time with the command* `openssl req -new -nodes -out my.csr -keyout my.key`.
>
> *You can have RapidSSL generate the private key and CSR for you. That introduces risks – the private key is no longer truly private.*
>
> *Be sure to store and back up the private key safely. If it's lost, your certificate will be worthless. If it's stolen, your site is no longer safe to use.*

Type the following command:

```
less my.csr
```

The Certificate Signing Request (the contents of *my.csr*) is shown. Select all of it and use *Edit>Copy* to copy it to the clipboard, then press *q* to quit the `less` command.

Paste the request into the box and continue with the application, making sure to choose an address at your domain where you can receive e-mails (e.g. postmaster@example.com).

When you receive the confirmation e-mail, click on the link that it contains and click *APPROVE*.

The signed certificate is e-mailed to you. Open the e-mail and select and copy the certificate text. Use the following command:

```
touch mysigned.crt
```

> *This creates an empty file called mysigned.crt.*

Open *mysigned.crt* for editing in Midnight Commander, then use

Edit>Copy to paste the certificate text into it. Press F2 to save your changes and F10 to exit the editor.

> *To limit the number of people with access to the root keys that browsers know about, it's now common for certificates to be signed with an intermediate key. The public part of this key is contained in an intermediate certificate that, in turn, is signed with a root key so that browsers can verify it.*
>
> *RapidSSL may tell you to download their intermediate file and save it as intermediate.crt. It makes more sense to save it as rapidssl.crt in case you have intermediate certificates from other Certificate Authorities too.*

Install *my.key* (if you haven't done this already) and *mysigned.crt* in Apache as described in Section 6.7.

Install any intermediate certificate file too, uncommenting (by removing the initial *#*) and setting *SSLCertificateChainFile* in */etc/apache2/sites-available/default-ssl* to point to it.

Reload Apache. Go to the https version of your website (e.g. https://example.com), preferably from a different computer. You should no longer see warnings about the certificate.

> *You want to see what other users do. You might have told your usual browser to ignore problems with the old certificate.*
>
> *There is a useful SSL site security checker at https://sslcheck.globalsign.com/en_GB. Open your web browser (on any computer), go there and enter the address (URL) of your server (e.g. www.example.com) in the box. The site is tested and, after a delay, a security rating and list of problems found is shown. Some of these may relate to your web server configuration and may indicate vulnerabilities that are only a concern if your users are entering or viewing very sensitive information.*

6.9 Install a key and certificate in Dovecot/Postfix

If you run the Dovecot and/or Postfix email services on your server, you can use a certificate to secure your users' client connections and partially secure the emails themselves. Provided that the domain name matches,

you can use the same certificate for this and your web server.

If there is an intermediate certificate, it and *my.crt* need to be concatenated. The following commands do this:

```
cd /etc/ssl/private
```

```
cat my.crt intermediate.crt >my.pem
```

> *This assumes you're using a certificate called my.crt obtained from an authority that uses an intermediate certificate that you've saved as intermediate.crt. If you've used different file names, use those instead.*

Dovecot uses the certificate to protect users' connections to the server, including making sure that their passwords cannot be seen by an eavesdropper. The key and certificate are set in */etc/postfix/main.cf*. Set *ssl_key_file* to point to your private key file (e.g. */etc/ssl/private/my.key*). Set *ssl_cert_file* to point to the certificate (e.g. */etc/ssl/private/my.crt*) or, if there is an intermediate certificate, to the combined file (e.g. */etc/ssl/private/my.pem*). Restart Dovecot with

```
/etc/init.d/dovecot restart
```

Postfix can use the certificate to encrypt transfers of emails to and from other servers on the internet. The key and certificate are set in */etc/postfix/main.cf*. Set *smtpd_tls_key_file* to point to your private key file (e.g. */etc/ssl/private/my.key*). Set *smtpd_tls_cert_file* to point to the certificate (e.g. */etc/ssl/private/my.crt*) or, if there is an intermediate certificate, to the combined file (e.g. */etc/ssl/private/my.pem*). Restart Postfix with

```
/etc/init.d/postfix restart
```

You can test that your server receives email securely at http://www.checktls.com/testreceiver.html.

> *Many other servers don't support encrypted connections: email sent to and from these won't be protected. Additionally, because your Postfix server doesn't know which other servers should be using encryption, someone can set up a man-in-the-middle attack. End-to-end encryption (e.g. using PGP, GPG or built-in email client encryption) should be used for any sensitive messages sent by public e-mail.*

Alphabetical Index

Apache....26
  reload....**27,** 29
attack....
  brute force....**7,** 10, 14
  keylogger....20
  known plaintext....6
  man-in-the-middle....**12,** 14, 25
  password....20, **21**
  phishing....20
  replay....6
  social engineering....22
certificate....18
  Certificate Authority....**18,** 19, 27
  Extended Validation....19
  intermediate....29
  RapidSSL....19, 27
  self-signed....19, 25, 27
  X509....25
code....3
cypher....3
  AES....**6,** 7, 12
  block....6
  DES....**6,** 7, 23
  El Gamal....15
  one-time pad....5
  RSA....10
  triple DES....23
  unbreakable....5
cyphertext....3
Debian....14
decrypt....3
Diffie Hellman....**16**
Dovecot....29
e-mail....26
encrypt....3
Enigma....4
Facebook....17
Google....17
government....

GCHQ....7
key escrow....8
NSA....7
GPG....**15,** 30
hash algorithm....13, 21, 23, 24
MD5....24
one-way....**13,** 21
SHA....**13,** 24
insider trading....8
Internet Key Exchange....**16**
John the Ripper....21
key....4
ephemeral....15
length....14
private....**10**
public....**10**
session....**12,** 18
symmetric....**10,** 12, 14, 18, 23
modulo arithmetic....**4,** 11
Perfect Forward Security....**16**
PGP....**14,** 30
plaintext....3
players....
Alice....3
Bob....3
Carol....10
Eve....3
Mal....5
Postfix....29
prime number....10, 24
factoring....11
rainbow table....21
RSA....**10,** 23, 25
modulus....**10,** 14, 24
private exponent....**11,** 24
public exponent....**10,** 24
salt....22
global....22
signature....11, 13
detached....13
Snowden....17

SSL....18
steganography....16
traffic analysis....16
Twitter....17
Ubuntu....14, 23
web browser....18, 19, 27
    Firefox....27
    Internet Explorer....27
    padlock symbol....18
web of trust....14