Safe online transactions

We rely on SSL and TLS every day, but what are they and how do they work, asks **Julian Bucknall**

Let me introduce some people who will help me talk about cryptography and SSL/TLS. First we have Alice and Bob. They live far apart and love communicating with each other, but because they want to keep their conversations secret, they encrypt all their messages. Eve is fascinated by these two and is continually eavesdropping on them, but that's all she does: listen in, trying to work out what they're talking about. Then there's Mallory. He not only listens and tries to work out what they're up to, but he's malevolent as well. He will alter their messages, delete them and substitute his messages for Alice's or Bob's, trying to fool them both that his messages originate from the partner. He is known as the man in the middle.

Back in the old days, Alice and Bob would use a shared key and an agreed-upon symmetric encryption algorithm. In 1981, the Data Encryption Standard (DES) was published publicly as a symmetric algorithm (that is, you encrypt and decrypt with the same key). Despite using what we might now think is a small key (only 56 bits), it took off and started the whole field of cryptanalysis.

Alice and Bob took to DES with abandon, but they ran into a problem: they needed a 56-bit key (preferably randomly generated) that they could share, but keep secret. Once the key was agreed on, all of their communications would be opaque to Eve and Mallory. There was just one problem – how could they agree on a key? Alice couldn't send a key to Bob, because both Eve and Mallory would see it as she'd have to send it unencrypted. Even worse, Mallory could substitute another key entirely and send that to Bob. After that, Mallory could intercept messages from Alice to Bob, decrypting them with the real key, reading them, then encrypting them with the fake key and sending them on. The same thing would happen on the return journey. Alice and Bob's messages would be nowhere near secure.

Shared keys

There was nothing for it: Alice and Bob would have to meet in person and devise a shared key, making sure that they couldn't be overheard by Eve or Mallory. Of course, if the shared key was ever disclosed or hacked, they'd have to go through the whole rigmarole of travelling to meet up and decide on a key again.

The most important thing to realise here is that the secret between Alice and Bob is the key. If the shared key was ever discovered, the totality of the communications between them would no longer be secure.

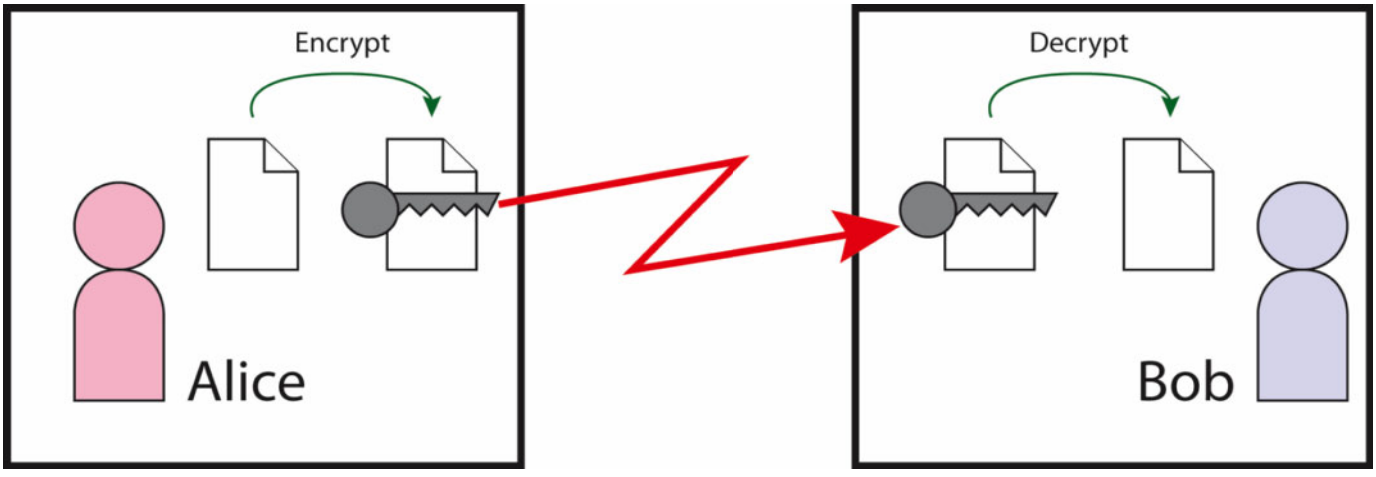
Then, two things happened: computers became fast enough to apply brute force decryption to messages encrypted with DES, and public key cryptography was invented.

With brute force decryption, you use a computer that tries every single key until one is found that decrypts the message (it assumes that the plaintext message is recognisable in some sense). When DES was first devised, PCs had only just entered the market and brute force cracking of a DES-encrypted message was infeasible. Nowadays, using a specially built computer, a DES 56-bit key can be discovered within a week on average.

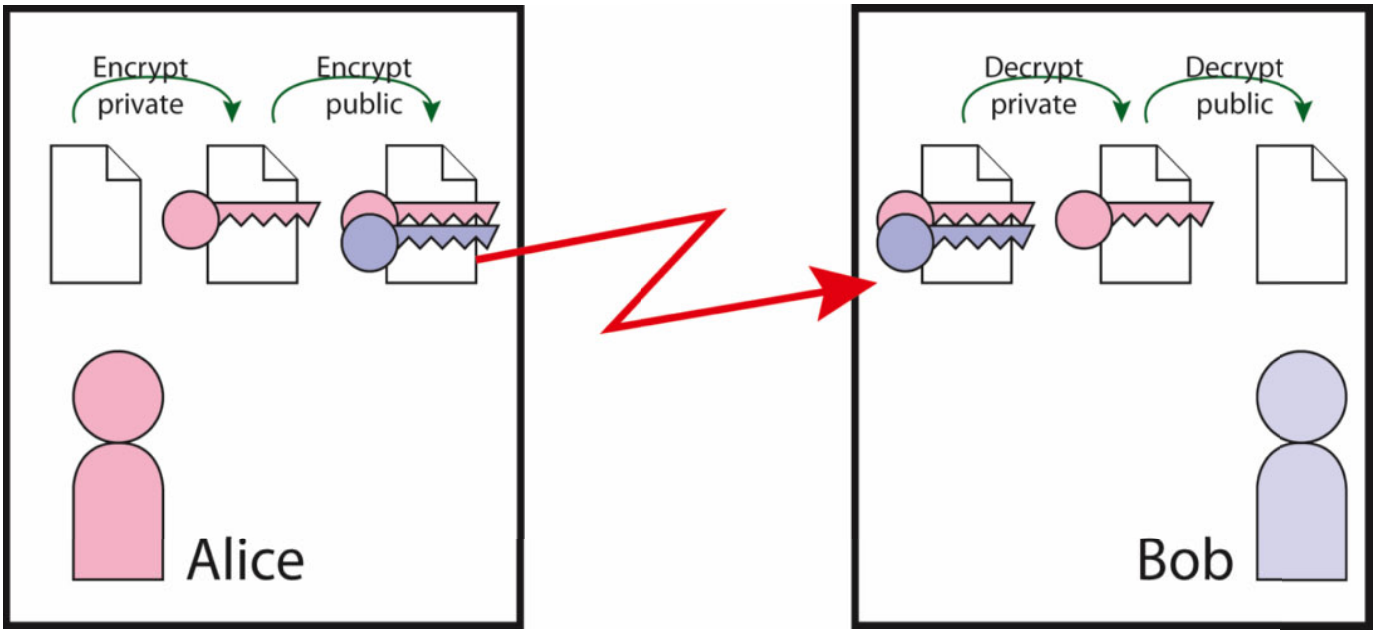
Standard DES has been supplanted with variations (triple-DES) and new algorithms (AES) with longer keys, but for Alice and Bob, the same old problem is still present: how to agree on and exchange a key securely.

Public key cryptography

With public key cryptography, things are different. Public key cryptosystems use two separate keys: a public key and a private key. The cryptosystem (the most famous one is RSA, named after its inventors Rivest, Shamir ▸



▲ **Secure communication with symmetric cipher. Alice and Bob use the same key to encrypt and decrypt messages, so agreeing on this key can be a problem.**



▲ **Secure communication with public key cipher. This system still leaves Alice and Bob with the problem of exchanging public keys securely.**

Spotlight on… Hacking SSL

Recently, Rizzo and Duong showed off a hack against SSL used with a block cipher. There are two main ways to encrypt with a block cipher: encrypt the blocks separately (known as electronic codebook, or ECB), or XOR the previous encrypted block with the current block and then encrypt it (known as cipher-block chaining, or CBC). The latter needs an initialisation vector (IV) – a random block to XOR with the first plaintext block. The block cipher SSL uses CBC with an IV.

When communicating with a server, data passes back and forth as separate packets. Does the IV for the first block of the next file or packet come from the last encrypted block of the previous file, or do we start afresh with a new IV? SSL uses the first option, which can lead to a small security hole. If the attacker is controlling one side of the channel over which the encrypted data is flowing, he can inject some specially constructed data into the stream using the previous IV. He guesses at the contents of the previous block, constructs his attack block with it, and if the resulting encrypted block is what he expects, he was correct.

But how does he gain control of one side? Rizzo and Duong used a Java applet that they injected into a web page through a cross-site hack. It isn't easy, but it shows that good security is hard. SSL was considered good security. Note though that although this hack has been proven, it doesn't mean that SSL is broken for most transactions on the web. Browser manufacturers are already fixing the problem. ■

and Adleman) uses special mathematical algorithms so that the encryption of a plaintext message and the decryption of that encrypted message use different keys. The keys are related mathematically, but knowing one doesn't really help you discover the other (the process involves the factorisation of a very large number into two very large prime numbers – an algorithm that with current mathematical knowledge would take an inordinate amount of time to calculate). Because there are different keys for encrypting and decrypting, these cryptosystems are known as asymmetric algorithms.

This is how Alice would encrypt a message to send to Bob with a public key algorithm. Both she and Bob have private/public key pairs, properly generated according to the algorithm they're using. Alice will encrypt the plaintext message with her private key (known only to her), and then encrypt the result of that with Bob's public key. She knows Bob's public key, because he publishes it (similarly she publishes her own public key). She then sends this twice-encrypted message to Bob.

Bob receives the encrypted message from Alice. He then decrypts the message with his private key (this key is a secret known only to him), and then decrypts the result of that with Alice's public key. If the result is legible, he knows a couple of things with certainty: only he could read it (neither Eve nor Mallory could, since only his private key could decrypt it), and Mallory couldn't have slipped in a fake message since the original message could only have been encrypted with Alice's private key. So everything is well, and he and Alice can communicate with abandon.

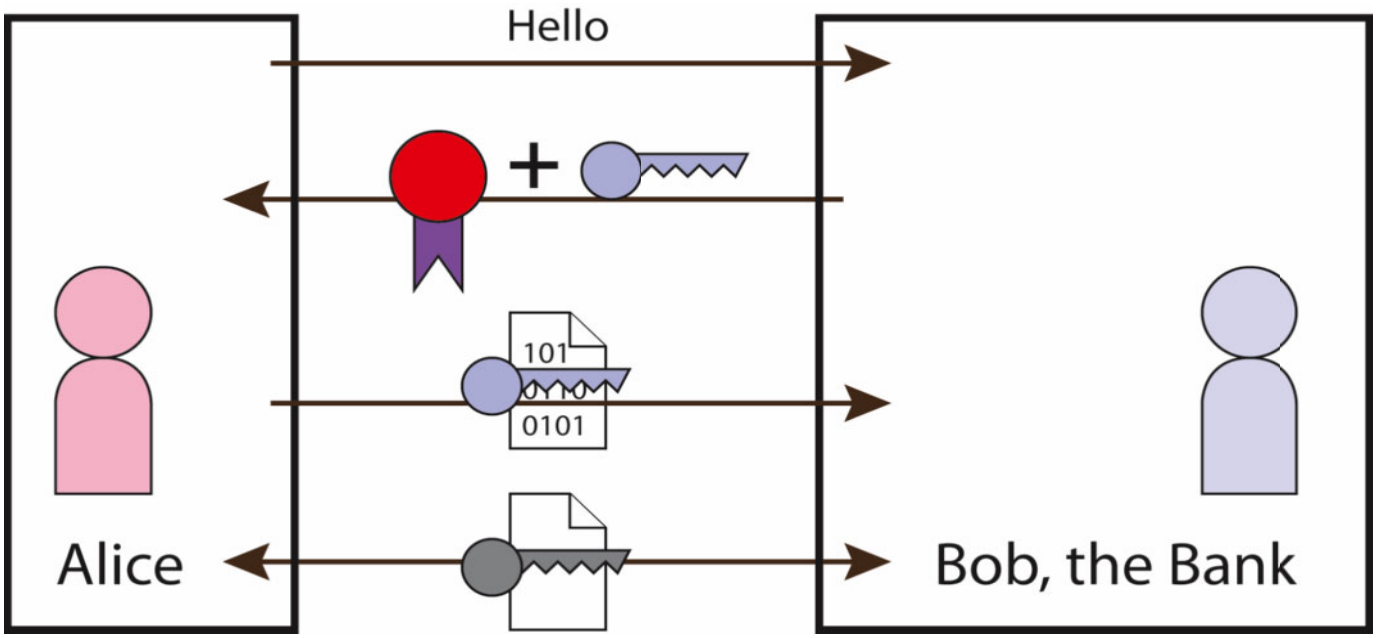
In fact, since public key cryptosystems are much slower at encrypting and decrypting than symmetric algorithms, in general only one message is sent using a public key cryptosystem: 'Here's a randomly generated key for a symmetric algorithm, let's both use that from now on.' All of a sudden, Alice and Bob's original problem with a symmetric encryption algorithm is removed: Alice just sends Bob a brand new 256-bit key encrypted using RSA in the manner I just described, and then they communicate using AES with that 256-bit key. They don't have to meet at all. Sounds great, but what's the flaw?

The flaw is this: how do Alice and Bob exchange their public keys securely? Alice can't send an unencrypted message to Bob containing her public key, because Mallory may intercept that message and substitute his own public key. (Ditto for Bob informing Alice of his public key.) If that did happen, Mallory would be in complete control of the message channel. Let's call the two key pairs that Mallory generates, fakeAlice and fakeBob; Alice thinks fakeBob is actually Bob, and Bob thinks fakeAlice is Alice. Suppose Alice sends a message to Bob. She encrypts it with her private key and then with fakeBob's public key and then sends it. Mallory gets it, decrypts it with the fakeBob's private key and with Alice's public key, and reads the message. He then encrypts a new message with fakeAlice's private key and Bob's public key, and sends it to Bob. Bob can decrypt it with his private key and fakeAlice's public key.

Suddenly it seems we're right back to square one: Alice and Bob still have to meet in order to exchange their public keys. We're no better off than we were before.

Certificate authorities

In practice, this problem is solved by one more level of indirection: the CA or certificate authority. A CA issues digital certificates that identify a particular person or entity and the public key used by that person or entity. In essence, a digital certificate is the name (usually a domain name) and the associated



▲ **Secure communication with SSL. Alice's browser verifies that Bob's certificate issigned by a trusted CA, then generates and encrypts a one-time public key.**

public key encrypted by the CA's private key. You can check the validity of a certificate by decrypting it with the CA's public key.

But hold on, you may be asking, how do Alice and Bob know the CA's public key? Can't Mallory just intercept this and replace with his own public key? Technically yes, but in practice the CA's public key is provided as a certificate with the browser or as part of the operating system. CA certificates are truly publicly published. You trust that these certificates are valid because they're delivered to you with your operating system or browser.

Once Alice and Bob buy their digital certificates from a particular CA, they can send them to each other with impunity, in essence by trusting the CA. Alice can check Bob's certificate (and discover his public key) by decrypting it with the CA's certificate, and vice versa. Once that's done, they can send each other secure messages ad infinitum.

Online banking

Now imagine that Alice is you, Bob is your bank, and you want to take a look at your accounts online and pay some bills. You certainly don't want Eve to see your account details, and definitely don't want Mallory to be fiddling with your transactions as you send then to your bank. Before RSA and public key systems, this would have been nigh on impossible: you would have had to securely agree on a large key with your bank. In fact, the bank would have had to agree on (and store) a random-looking key for all of its customers and keep them safe from prying eyes. The bank would have a nigh-on impossible task keeping the world's Eves and Mallorys from joining as employees and accessing all those private keys.

But with public key cryptosystems, this all becomes feasible. It is the basis of SSL (Secure Socket Layer) and TLS (Transport Layer Security). The latter is the newer version of the former, but everyone still uses the term SSL – although it does look a little different. The first problem is that we normal people don't (usually) have a private/public key pair and a digital certificate that proves who we are (for a start, certificates are very expensive), so we have to approach things differently.

SSL explained

When you visit a bank's website, the bank's server will automatically redirect you to its secure site using the HTTPS protocol before you can log in. This results in your browser and the bank's site negotiating a secure channel using SSL. This negotiation goes a little like this (note that I've simplified it greatly).

The browser sends a message stating what the latest version of SSL it can support and a list of symmetric algorithms it can use.

The web server sends back a message with the version of SSL and the algorithm that will be used. It sends its certificate as well.

The client verifies the certificate using the known certificates that came with the browser; in other words, it checks that it has been signed by a trusted CA and that it hasn't expired.

If the certificate is valid, the browser generates a one-time key for the session, encrypts it with the server's public key (it's part of the certificate), and sends it to the server.

The server decrypts the key, then uses that key together with the agreed symmetric algorithm for the rest of the session.

Let's take stock. Your browser is certain that the server is who it says it is (your browser is trying to access YourBank.com, and the certificate says it's valid for YourBank.com – and the CA agrees). The browser has generated a cryptographic key that will be used for one time only: this particular session. It'll be thrown away after you log out. The key was sent encrypted with YourBank.com's public key, which only YourBank.com can decrypt with its private key. There are a couple of other messages sent that check your browser and the web server have agreed on the same key (if anything went wrong, the session is dropped).

Once YourBank.com has presented you with a login screen (which will be sent over HTTPS, if the bank knows what it's doing) and you've filled it in, it'll know who you are. Your id and password will have been sent encrypted over the secure channel that you've both established. Eve and Mallory are completely out of the loop. PCP

DigiNotar

DigiNotar was a certificate authority in the Netherlands. On July 19 2011, hackers accessed DigiNotar's systems. They created around 30 fake certificates for well-known domains, including google.com, microsoft.com, and mozilla.org.

Consider what they could do with a certificate for google.com. We'll assume they are called Mallory and have governmental powers so they can monitor all internet traffic. Alice tries to connect to Gmail, but would in fact be setting up an SSL session with Mallory's server (Mallory's certificate says it is google.com, and this Dutch CA proves it). Mallory's server would then set up a session with the real Gmail so he became the man in the middle. At that point Mallory could read all of Alice's emails without her being aware of it.

It's thought that Mallory in this case was the Iranian government and that it was trying to gain information about dissidents. The solution, once the bogus certificates were discovered, was to renounce DigiNotar as a CA. Its trust was revoked and all certificates it issued were made invalid.

Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft and is now CTO for Developer Express. He discussed the latest innovations in AI in issue 314.
feedback@pcplus.co.uk