Forward Secrecy in Java

By Andy Brodie ([andy.brodie@worldpay.com](mailto:andy.brodie@worldpay.com))

# Introduction

This short paper presents an overview of forward secrecy (FS) in two parts. The first part describes the background and motivation for FS; and the second describes how to use FS in Java applications using the Java Secure Sockets Extensions (JSSE).

This document is designed for readers that have a little limited understanding of security, and need or want to increase their understanding. Such people, like myself, find themselves frequently confused or frustrated by the many use of interchangeable or non-intuitive terms or use of mathematical vocabularies that use unfamiliar words for very simple ideas.

Forward Secrecy is sometimes referred to as *Perfect* Forward Secrecy. Several cryptographers have recommended against include the word “perfect” because it gives a false impression that there will never be a weakness or imperfection in this technique. Therefore, this paper refers to Forward Secrecy only.

# Problem Statement

Modern web browsers and applications commonly use the TLS[[1]](#footnote-1) protocol to ensure that communication between systems over a network is *secure*. Makingcommunication secure has many different aspects, such as authentication, non-repudiation, integrity, confidentiality and so on. FS is primary concerned with confidentiality; i.e. preventing any unwanted disclosure of information.

## Bob, Alice and Eve

We’ll refers to the standard[[2]](#footnote-2) actors Alice, Bob and Eve to describe the interactions between parties involved in the communication.

Alice wants to have a series of conversations, that is exchange electronic messages, with Bob. Alice and Bob want all the conversations between them to be private (a type of confidentiality). More specifically, no-one apart from Alice and Bob should ever be able to discover the contents of their conversations.

Eve is Alice and Bob’s antagonist, an eaves-dropper who listens in on the conversation between Alice and Bob. If Alice and Bob communicate with each other over the Internet, then Eve could be anywhere on the Internet where network traffic between Alice’s browser and Bob’s server flows.

Therefore, in order for Alice and Bob’s conversation to be private, it needs to be encrypted in such a way that only Bob can understand what Alice is saying (and vice versa). This is pretty much the only way to do it if Alice and Bob are communicating in a public forum such as the Internet. The bits and bytes of data that makes up the conversation are encrypted so it cannot be read by anyone other than Bob or Alice.

### Encryption using Symmetric and Asymmetric Keys

Encryption and decryption is typically done with keys, which are simply numbers which are applied to mathematical functions to either encrypt or decrypt data.

Depending on the mathematical function, keys are generally either symmetric or asymmetric:

1. Symmetric[[3]](#footnote-3) encryption is where Alice and Bob share a secret key. The same key is used to encrypt messages as is used to decrypt messages. Exactly like a standard lock you have on a door. Alice and Bob can use the same key to lock or unlock the door.
2. Asymmetric encryption[[4]](#footnote-4) is slightly more complicated. There are two keys, referred to as a key pair. Anything encrypted by one key can be decrypted by the other key. However, given one key, it is very difficult to work out what the other key is.

The biggest application of asymmetric encryption techniques is Public Key cryptography. Public key cryptography simply states that one of the keys is public (i.e. shared to anyone who wants it), and the other is private. This, of course limits private communication to a single direction, because if you encrypt with the *private* key then anyone with the *public* key (which should be everyone) can decrypt it.

A real world analogy would be using a padlock. Bob has a padlock with one key. He knows Alice wants to send him a message, so he mails the open padlock to Alice. Alice puts her information in a box, and locks it with the padlock and mails it to Bob. If Eve intercepts the box, she can’t open it, because only Bob has the key.

Ultimately, therefore Eve’s goal is to obtain a key that allows her to decrypt all of Alice and Bob’s conversations.

## How can Eve get the key?

Eve could try to obtain the necessary key by attacking in three ways:

1. Working out via a weakness in the cipher mechanism or the means by which the key was created;
2. Guessing (a *brute force attack*), by trying every possible key until the right one was found; or
3. Obtaining the key by methods such as social engineering or gaining physical access to Alice and Bob’s systems.

For security to be effective, all three of these attacks should be unfeasibly difficult for Eve.

Finding, exploiting and protecting against weaknesses in the cipher mechanism is a multi-billion-dollar industry and the subject of intense academic scrutiny. Neither Alice, Bob nor Eve are cryptographers or mathematicians, so they just have to rely on faith that those guys know what they’re doing and will tell them what they need to do.

Brute force attacks, or guessing the key from all the potential possibilities is also designed by the clever people above to be infeasible. The bigger the key, the more possibilities there are. However, the more powerful computers become the more keys they can try in a period of time. So over time the size of keys generally increases[[5]](#footnote-5).

The third type is usually the weakest point for Eve to attack. Hacking Bob or Alice’s computer or using social engineering techniques to obtain the key, for example: “Hi, I’m Eve. I’m your local security inspector, could I see your keys to make sure they’re suitably secure please?”; “Tell me the key or you’ll be prosecuted under the RIPA 2000 act”; or even “Give me the key, or the bunny gets it.”

Many kinds of attacks on Alice and Bob’s confidentiality are designed with the goal of Eve listening in *real time* to the conversation (as it happens). I.e. the effects of security are rendered entirely useless and it is as if no security was present at all.

However, in other cases, Eve doesn’t *need* to decode the conversation in real time. Eve may only desire that at some point in the future she can read what Alice and Bob talked about; maybe months or years after the actual conversation happened.

Therefore, rather than trying to decrypt the conversation is real-time as it flows past her, Eve instead records every aspect of Alice and Bob’s encrypted conversations, stores them for safe-keeping and working on her attacks in the hope that one day, she’ll get the key and find out what Alice and Bob have been saying about her!

## How TLS Delivers Confidentiality

In order to understand what Forward Secrecy is doing and how it’s doing it, we need to understand how TLS works at a basic level.

TLS provides both authentication and confidentiality for conversations. Usually, the client (Alice) authenticates the server (Bob), so that she’s confident that it’s really Bob who’s talking to her. TLS supports client authentication, but it’s rarely used, instead relying on usernames and passwords. For example: you know you’re using Amazon on your browser because TLS authenticates Amazon. However, you then authenticate yourself to Amazon by logging in with a username and password. This second stage is not part of TLS.

So, Alice and Bob have two goals for their conversation:

1. Alice needs to be sure it’s Bob she’s talking to.
2. Alice and Bob need to agree how to encrypt messages that flow between them.

TLS does not prescribe *exactly* how Alice and Bob achieve this. TLS just states the kinds of messages that Alice and Bob must exchange to achieve this and allows them to “negotiate” a mutually agreed way to do so.

TLS negotiates, amongst other things, how Alice and Bob will:

1. Authenticate each other.
2. Exchange a key to encrypt all the messages with.

The negotiation is very simple: Alice tells Bob all the different combinations of algorithms she supports, in order of preference, and Bob responds with which one he’s going to use. Bob may or may not take in to account Alice’s order of preference. If Alice is unhappy with Bob’s choice, she can terminate the conversation before it starts. If Bob isn’t happy with the list that Alice offers, he can also terminate the conversation before it starts.

Rather than letting servers and clients arbitrarily mix and match algorithms and their configuration, TLS defines a set of 37 cipher suites (T. Dierks, 2014) which describe combinations of algorithms and basic configuration of those algorithms to use to meet different security requirements for conversations.

## Certificates and Keys in Common TLS Cipher Suites

TLS uses certificates to perform authentication. A certificate is simply some information about a *subject* (E.g. Bob) a public key and some other stuff, all of which are signed by a *certificate authority* that is trusted by both Bob and Alice.

When Alice receives the certificate from Bob, she checks the certificate and decides whether she trusts it.

Trust comes from a *certificate* *authority* that both Alice and Bob trust before they start their conversation. Bob’s certificate was issued by the certificate authority when they were satisfied that it really was Bob that was asking for one. Certificate Authorities take time and charge for issuing certificates, so Bob can’t have a new certificate for every conversation. Once issued, certificates typically last for months or years. So, the public key contained in the certificate, along with its corresponding private key, be thought of as static.

This is the root of the problem of forward secrecy: if Alice and Bob rely on Bob’s private key to protect all their conversations, then if Eve manages to obtain Bob’s private key at any point in future, she has access to *all* the recorded conversations between Alice and Bob.

## Forward Secrecy in TLS

Forward secrecy limits the damage of a private key exposure by demanding that:

1. The transmission of the pre-master secret is protected by a different asymmetric key pair for each session.
2. That asymmetric key pair is unpredictable.

The second property, making a key unpredictable, is not easy. But fortunately some very clever people have designed some cryptographically secure pseudorandom random number generators[[6]](#footnote-6) (CSPRNGs). How these algorithms work is out of scope here; so Alice and Bob just have to have faith that those clever academic and professional cryptographers and mathematicians know what they’re talking about. The asymmetric key pair is then generated using random numbers.

Forward secrecy is therefore very simple and straightforward: instead of using Bob’s private key to encrypt handshake data, Bob creates a new public/private key pair for *every conversation* and sends Alice the public key to use to encrypt handshake information, such as a pre-master and master secret.

This is an *ephemeral* key pair and has two properties:

1. It is never persisted outside volatile RAM.
2. It is discarded as soon as the handshake is completed.

This means that Eve has not only an incredibly difficult job to obtain the private key, but even if she did manage to obtain a private key, it will only work for a single session (conversation) between Alice and Bob.

# Forward Secrecy in Java

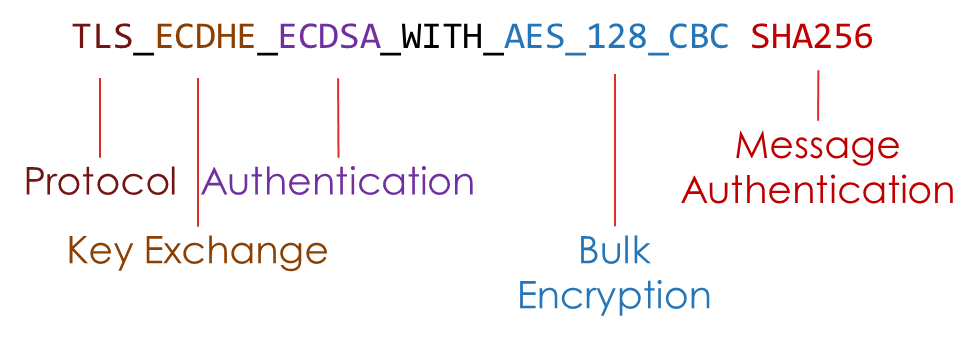
Now we’ve seen the theory and know what needs to happen, we will now look at how we can do this for real using JSSE in Java 8.

## Cipher Suites

Java(TM) SE Runtime Environment (build 1.8.0\_60-b27) ships with the Sun (Oracle) cryptographic provider. Cryptographic providers are implementations of various aspects of security-related code, or “Engine” classes. These engines implement different ciphers, message digests, key exchange algorithms, and so on (Oracle, 2015).

Sitting on top of these providers is the Java Secure Sockets Extensions (JSSE). These provide an API for implementing the TLS (and SSL) protocols in applications.

JSSE uses the available cryptographic providers to identify a set of cipher suites that can be used to implement TLS. Cipher suites are identified by long string names, which consist of the different component pluggable parts of the TLS protocol:



|  |  |  |
| --- | --- | --- |
| Term | Description | Examples |
| Protocol | The protocol that will be implemented. Note that the version is not included. | SSL, TLS. |
| Key Exchange | How a symmetric key is agreed upon and exchanged between Alice and Bob. | DH, DHE, ECDH, ECDHE, RSA, KRB5. |
| Authentication | How the two parties having a conversation authenticate each other. | DSS, ECDSA, RSA, NULL. |
| Bulk Encryption | What algorithm is used to protect the main body of the conversation. Our secret key is used by this algorithm to encrypt the messages between Alice and Bob. | 3DES\_EDE\_CBC, RC4\_128, AES\_128\_CBC, NULL. |
| Message Authentication | How parts of the main body of the conversation are signed to prevent undetected tampering. | MD5, SHA, SHA256. |

Forward secrecy is controlled by the “Key Exchange” algorithm used:

|  |  |  |
| --- | --- | --- |
| Key Exchange Algorithm | Acronym Expansion | Provides Forward Secrecy? |
| DH | Diffie-Hellman | No |
| DHE | Diffie-Hellman **Ephemeral** | Yes |
| ECDH | Elliptic Curve Diffie-Hellman | No |
| ECDHE | Elliptic Curve Diffie-Hellman **Ephemeral** | Yes |
| RSA | Rivest, Shamir and Adleman | No |
| KRB5 | Kerberos | No |

## Java Secure Sockets

JSSE (more or less) allows you to take an existing program that communicates using the Socket and ServerSocket classes and wrap the TLS protocol around it. This allows you to take a program that was once communicating in plaintext, and convert it to use TLS without modifying the original code.

ServerSocket and Socket instances are created by ServerSocketFactory and SocketFactory instances. The concrete implementations of these abstract classes create plain sockets, but JSSE provides SSLServerSocketFactory and SSLSocketFactory classes to use TLS.

## Demo Application

A demonstration application is available for readers to review and experiment with the Java JCA and JSSE features.

The application consists of a simple server and client application: TimeServer and TimeClient. TimeServer opens up a listening socket and waits for TimeClient to connect to it. TimeServer sends the current date and time back to TimeClient as a string, closes the socket and terminates. TimeClient prints out the received string and terminates.

Neither server nor client have any dependency on JSSE, so both can be run either using “plain” sockets or using TLS.

A set of command line programs are available that perform various different functions.

The code for the application is available on GitHub to download.

To build and run the code, you’ll need:

1. Java 1.8, build 1.8.0\_60-b27 or compatible.
2. Apache Maven 3.3.3 or compatible.

### Code Organisation

|  |  |
| --- | --- |
| Package | Contents |
| com.worldpay.fsdemoapp | Classes that deal with the JCA and JSSE to configure TLS. |
| com.worldpay.fsdemoapp.businesslogic | The sample server and client program that are ignorant of TLS. |
| com.worldpay.fsdemoapp.programs | Various command line programs that demonstrate different scenarios. |

# Word Limit

300 words in 2 minutes mean max. 2000 words.

# Appendices

## Confusion

Several resources (Pillai, 2013) (Bernat, 2011) that I read during the creation of this document confuse two concepts:

1. That Diffie-Hellman Key exchange can be done in the clear.
2. DH does *not* automatically provide forward secrecy, nor is it requiredfor forward secret.

## Diffie-Hellman Key Exchange

The Diffie-Hellman algorithm for key exchange (Hellman, 1976) dates back to 1976, and provides a way for Alice and Bob to establish a shared, secret key without ever revealing the key in a way that Eve could work it out.

The importance of this breakthrough in cryptography can be appreciated better if you consider a human equivalent example: imagine Alice meets Bob in a coffee shop, they’ve never met before. Alice and Bob sit opposite each other, across a small table, talking in loud voices[[7]](#footnote-7). However, before Alice and Bob even entered the coffee shop, Eve was already in there, listening to conversations going on around her. She sees Alice and Bob enter and sit within her earshot. How on earth could Alice and Bob, who have never communicated before, possibly agree a way to encrypt a conversation between them such that Eve *cannot* eaves-drop on the later conversation, without resorting to non-verbal communication?

This is the problem that Whitfield Diffie, Martin Hellman and Ralph Merkle[[8]](#footnote-8) solved[[9]](#footnote-9).

## Difference between RSA and DH

RSA is a cryptosystem that includes asymmetric encryption and digital signature algorithms. DH is an algorithm for key exchange only, so they’re not directly comparable as a whole. However, asymmetric encryption can be considered very similar to key exchange. They both achieve the same thing, just through different means (i.e. that a shared key is established that can be used for symmetric encryption).

Just considering the asymmetric encryption part of RSA and key exchange part of DH as distinct because:

|  |  |
| --- | --- |
| RSA | Based on the difficulty of integer factorisation. |
| DH | Based on the discrete logarithm problem. |

Additionally, the DH algorithm is much faster for computers to generate the exchanged key than RSA. For this reason, it is more commonly used for ephemeral key generation.

Asymmetric encryption is computationally “hard”[[10]](#footnote-10). Therefore, the TLS protocol, wanting to be efficient, says that Alice and Bob will use asymmetric encryption to agree a symmetric key (known as a “shared key” or “master secret”) which is then used to encrypt the messages that flow between Alice and Bob.

And here’s the problem: if all of the shared keys are exchanged between Alice and Bob are protected by a same unchanging public key, then all Eve needs is that one private key to decrypt the messages containing the secret key, and achieve her goal of reading all the messages that Alice and Bob ever exchanged.

# Appending – Pre-Master Secrets, Master Secrets and Symmetric Keys.

Confusingly, each of the items in the heading above are different and none of them relate to the Bob’s public/private key pair. However, they are required for both integrity, interoperability and performance in TLS.

The most obvious question is: “Bob already has a public/private key pair, so why not use that?”. Because Eve has access to Bob’s public key, anything Bob encrypts with his private key, Eve can decrypt it with the public key. So whilst this works ok for Alice talking to Bob, it doesn’t work the other way around. You could get around this by Alice having her own certificate, but asymmetric key encryption is pretty CPU intensive, therefore slow to encrypt and decrypt messages. This might not be an issue for a single conversation, but for high-volume sites such as Amazon or Facebook, this is really important.

For brevity, we’ll short-circuit the discussion on exactly what pre-master and master secrets are and skip directly to the goal: Alice and Bob are trying to agree a symmetric key to encrypt and decrypt messages between themselves. The symmetric key must be protected from Eve finding it out. The pre-master secret and master secret are used to generate this key.

Note that changing the symmetric key (something that TLS does regularly) won’t help here, because every time a new key is agreed, it’s shared between Alice and Bob encrypted by the same private key.

This symmetric key is only used for the actual conversation between Alice and Bob, parts of the TLS *handshake*, which only occurs once per conversation, use the public/private key to encrypt communication from Alice to Bob.

So, our problem exists whilst the pre-master and master secrets and shared between Alice and Bob using a connection encrypted with Bob’s public key.

This is, unfortunately, how all the cipher suites in SSL/TLS worked before the need for forward secrecy was identified.

## RSA, Diffie-Hellman and Terminology

Before we go any further, the term “RSA” is often used to refer to the process authentication and key agreement process as a whole, as well as to refer to individual parts. Diffie-Hellman is only a key agreement algorithm.

*Key Agreement* is used as an equivalent term for *Key Exchange*. I prefer Key Agreement because *exchange* implies that the key is being sent (encrypted) from one party to another, where as key *agreement* does not imply this.

The detail of how DH works (see Appendix) make the difference apparent because, using DH, Bob and Alice can *agree* on a key but the key is never *exchanged* (sent) to Alice and Bob. I.e. DH key agreement can actually occur in plaintext and stillbe secure[[11]](#footnote-11).

# Bibliography

Alfred J. Menezes, P. C. (2011, July 8). *Handbook of Applied Cryptography*. Retrieved from Centre for Applied Cryptographic Research: http://cacr.uwaterloo.ca/hac/

Bernat, V. (2011). *SSL/TLS & Perfect Forward Secrecy | Vincent Bernat*. Retrieved from Disruptive Ninja: http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html

Dictionary.com. (n.d.). *Ephemeral Definition*. Retrieved from Dictionary.com: http://dictionary.reference.com/browse/ephemeral?s=t

Dierks, T. (2014, May 23). *Tim Dierks: Security Standards and Name Changes in the Browser Wars*. Retrieved from Tim Dierks : http://tim.dierks.org/2014/05/security-standards-and-name-changes-in.html

Hellman, W. D. (1976, November 6). *New Directions in Cryptography*. Retrieved from Stanford University: http://www-ee.stanford.edu/~hellman/publications/24.pdf

Helme, S. (2014, May 10). *Perfect Forward Secrecy - An Introduction*. Retrieved from Scott Helme: https://scotthelme.co.uk/perfect-forward-secrecy/

Lu, S. (2013, June 30). *Can someone explain a little better what exactly is accomplished by generation of DH parameters?* Retrieved from Information Security Stack Exchange: http://security.stackexchange.com/questions/38206/can-someone-explain-a-little-better-what-exactly-is-accomplished-by-generation-o/38252#38252

Mortier, R. (2013, 12 2). *Explaining perfect forward secrecy*. Retrieved from Phys.org: http://phys.org/news/2013-12-secrecy.html

Moser, J. (2009, June 10). *The First Few Milliseconds of an HTTPS Connection*. Retrieved from Moserware.com: http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html

Moser, J. (2009, June 10). *The First Few Milliseconds of an HTTPS Connection*. Retrieved from Moserware: http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html

Mozilla. (2015, Aug 28). *Security/Server Side TLS*. Retrieved from Security/Server Side TLS / MozillaWiki: https://wiki.mozilla.org/Security/Server\_Side\_TLS

Oracle. (2015). *Java Cryptography Architecture (JCA) Reference Architecture*. Retrieved from Java Documentation: http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

Oracle. (2015). *Java Cryptography Architecture (JCA) Reference Guide*. Retrieved from Oracle Documentation: http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#Provider

Oracle. (2015). *Java Secure Socket Extension (JSSE) Reference Guide*. Retrieved from Java Documentation: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html

Oracle. (2015). *Standard Algorithm Name Documentation*. Retrieved from Java Documentation: https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html

Palmgren, K. (2000, July). *Diffie-Hellman Key Exchange – A Non-Mathematician’s Explanation*. Retrieved from SecurityPortal.com: https://docs.google.com/viewer?a=v&pid=sites&srcid=bmV0aXAuY29tfGhvbWV8Z3g6NTA2NTM0YmNhZjRhZDYzZQ

Pillai, S. (2013, 12 14). *What is Perfect Forward Secrecy and its impact on SSL (HTTPS)*. Retrieved from /Root.in: http://www.slashroot.in/what-perfect-forward-secrecy-and-its-impact-ssl-https

Pornin, T. (2011, November 7). *What is the purpose of four different secrets shared by client and server in SSL/TLS?* Retrieved from Cryptography Stack Exchange: http://crypto.stackexchange.com/questions/1139/what-is-the-purpose-of-four-different-secrets-shared-by-client-and-server-in-ssl

Pornin, T. (2012, 09 29). *How does SSL/TLS work?* Retrieved from Security Stack Exchange: http://security.stackexchange.com/questions/20803/how-does-ssl-tls-work

Pornin, T. (2012, September 29). *How does SSL/TLS Work?* Retrieved from Information Security Stack Exchange: http://security.stackexchange.com/questions/20803/how-does-ssl-tls-work

Pornin, T. (2013, May 8). *Is there any particular reason to use Diffie-Hellman over RSA for key exchange?* Retrieved from Stackexchange: http://security.stackexchange.com/questions/35471/is-there-any-particular-reason-to-use-diffie-hellman-over-rsa-for-key-exchange

Pornin, T. (2013, May 8). *Is there any particular reason to use Diffie-Hellman over RSA for key exchange?* Retrieved from Information Security Stack Exchange: http://security.stackexchange.com/questions/35471/is-there-any-particular-reason-to-use-diffie-hellman-over-rsa-for-key-exchange

R.L. Rivest, A. S. (1977, September 1). *A Method for Obtaining Digital Signatures and Public Key Cryptosystems.* Retrieved from MIT: https://people.csail.mit.edu/rivest/Rsapaper.pdf

Ristić, I. (2014, March 11). *Ivan Ristić: Significant SSL/TLS Improvements in Java 8*. Retrieved from Blog: Ivan Ristić: http://blog.ivanristic.com/2014/03/ssl-tls-improvements-in-java-8.html

T. Dierks, E. R. (2014, April 17). *The Transport Layer Security (TLS) Protocol Version 1.3*. Retrieved from tools.ietf.org: https://tools.ietf.org/html/draft-ietf-tls-rfc5246-bis-00

Various. (2015, 09 12). *Forward Secrecy - Wikipedia, the free encyclopedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Forward\_secrecy

1. SSL (Secure Sockets Layer) was renamed to TLS (Transport Layer Security) in 1999 and is published as a set of RFCs. The name was changed for political reasons (some claim legal issues with Netscape), but TLS 1.0 can be thought of as SSL 3.1. Ref: http://tim.dierks.org/2014/05/security-standards-and-name-changes-in.html. [↑](#footnote-ref-1)
2. Alice, Bob and others have been ubiquitous for describing actors in secure systems since 1977 when Ron Rivest published information on the RSA (Rivest, Shamir and Adleman) cryptosystem using these names as examples. (R.L. Rivest, 1977) [↑](#footnote-ref-2)
3. Examples of widely used algorithms are DES (Data Encryption Standard), 3-DES (Triple DES), RC2, RC4, AES (Advanced Encryption Standard) and Blowfish. [↑](#footnote-ref-3)
4. Examples of widely used asymmetric key generation algorithms are RSA, DH and DSA. [↑](#footnote-ref-4)
5. We’re not going to discuss quantum computing which, if the theory is made in to reality, could undermine this assumption. We will refer to elliptic curves a bit later, which use shorter keys without compromising security. [↑](#footnote-ref-5)
6. Examples of widely used CSPRNGs are FIPS 186-2, NIST SP 800-90A. [↑](#footnote-ref-6)
7. Alice and Bob are Americans. [↑](#footnote-ref-7)
8. Diffie and Hellman wrote and published the 1976 paper describing the key exchange protocol. In 2002 Hellman suggested that as they had built so much on, and used pioneering work by Ralph Merkle’s, it was only right that he should be equally credited. [↑](#footnote-ref-8)
9. They weren’t the first, though. GCHQ researchers Ellis, Cocks and Williamson discovered the technique in 1975. However, their work was held as secret until 1997 until the classification expired. For this, they were presented with the 100th IEEE Milestone award. <http://theinstitute.ieee.org/technology-focus/technology-history/cryptography-breakthrough-is-100th-milestone624>. [↑](#footnote-ref-9)
10. This is due to the mathematical complexity of the algorithms as well as the key lengths typically used. Use a really long asymmetric key to encrypt the message containing the shorter symmetric key and you get the best of both worlds. [↑](#footnote-ref-10)
11. The author considers this to be one of the most awesome things ever. [↑](#footnote-ref-11)