Securing Communications with the Diffie-Hellman Key Exchange

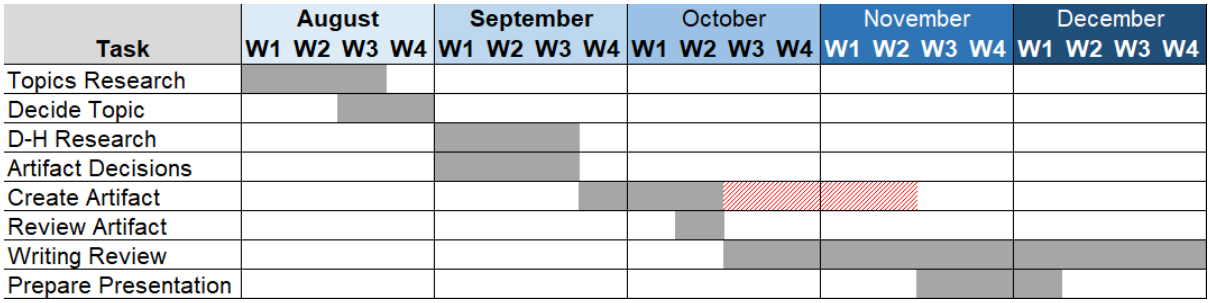
Freddy Heppell

Table of Contents

Planning

Gantt Chart

To assist with scheduling of the project, a Gantt chart was created. This chart details the process of the project from researching initial ideas to the writing of the review document.



The red hatched area indicates an overrunning of a task.

Source Evaluation

The majority of sources used are research papers published in respected journals, in many cases by the original inventors of the technologies. Other sources include textbooks, official documentation and magazine articles. Below, the reliability of some of the key sources is evaluated. All sources are fully listed in the bibliography (page 9).

Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice (2015)

This paper describes how it is possible to break the cryptography in the Diffie-Hellman key exchange if it is performed improperly, and how this can be rectified. It also details the real-world implications of these flaws and what percentage of popular websites are vulnerable. The study was conducted by reputable organisations including Microsoft, INRIA (The French Institute for Research in Computer Science and Automation) and the University of Michigan. The study was conducted in accordance with security research principles, including the standard 90-day nondisclosure period so that companies identified as being vulnerable have time to rectify the vulnerabilities. I used this source to ensure that my program followed the correct security principles by seeing how some common implementations fail.

New Diffie-Hellman Speed Records (2006)

This paper was written by Daniel J. Bernstein, a computer science professor at Eindhoven University of Technology. The paper describes how his implementation (called Curve25519) of the Diffie-Hellman procedure works, and why it is faster than existing implementations. The paper was published in the proceedings of the 2006 *9th Annual Conference on Theory and Practice in Public-Key Cryptography*, a conference organised by the International Association for Cryptologic Research. I did not use this source's methods in the development of my artefact, however I did use it to compare the speed of the program to a real world value.

Keeping Secrets (2014)

This is an article in the Stanford Alumni magazine about the legal dispute (dubbed the 'Crypto Wars') surrounding the presentation of the Diffie-Hellman algorithm at the International Symposium of Information Theory at Cornell University in October 1977. The article is written from the perspectives of both the Stanford cryptographers and the NSA and covers both the reasons in favour of the NSA's restrictions and in favour of the free speech rights of the Stanford cryptographers. Whilst this source was useful to learn about the discovery of Diffie-Hellman and the political climate at the time, I did not directly use this information.

More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) (2003)

This source is an RFC (*Request for Comments)* published by the IETF (*Internet Engineering Task Force*), an organisation that creates voluntary standards for internet technologies. Whilst, as of yet the standard has not been approved, it still provides constants that are representative of real-world sizes. The RFC lists MODP groups (a large prime and a small modulus) for various prime sizes. The intention of this is to allow browsers and other users of the algorithm to quickly decide on the numerical constants to use. The procedure through which these MODP groups are specified is considered to be trustworthy because of its usage of *"Nothing-up-my-sleeve numbers"*, that is they define the MODP groups based on known, unchangeable values (in this case π, although e and $\sqrt{2}$ are also common).

Secure Communications Over Insecure Channels (1978, received 1975)

This paper was written by computer scientist Ralph Merkle. It was the first research paper to identify the possibility of a key-exchange system that did not require an entirely secure channel. The Diffie-Hellman algorithm was based on this research, and in many cases Merkle is credited as the third inventor of the algorithm. In this paper he defined a precursor to the Diffie-Hellman algorithm, *Merkle's Puzzles*. Although the paper was published after *New Directions in Cryptography*, it was received prior to the publication but held due to the unknown legal status of publishing cryptographic papers at the time (Corrigan-Gibbs, 2014).

Understanding Cryptography (2010)

This is a cryptography textbook, intended for undergraduate courses. It provides background information on common concepts (for example the *XOR* operator which is commonly used in cryptography) and detailed information and proofs of specific algorithms. I used this source extensively developing the artefact as it clearly states the process through which the algorithm is performed and considerations that have to be made to ensure it is secure.

The Java Language Environment (1997) & *Java Platform Standard Edition 8 Documentation*

These documents are produced by Sun and Oracle respectively, the two companies that created and later maintained the Java language. The former is part of a whitepaper that describes the way in which the Java language is designed to operate. The latter is the official documentation for the Java 8 language. The documentation was my main source for writing the artefact as it describes the way in which functions must be interacted with (for example what data must be passed to them, and what data they will return).

Essay

Background Information

Diffie-Hellman Key Exchange

The Diffie-Hellman Key Exchange is an algorithm developed by Whitfield Diffie and Martin Hellman, based on work by Ralph Merkle (Diffie & Hellman, 1976).

Two computers, **A** and **B** wish to communicate securely. They publicly agree on a modulus p and a base g, where $2 \leq g \leq (p-2)$. Each of **A** and **B** privately generate a random secret, a and b respectively. **A** performs $A = g^a \bmod p$ and sends this to **B**. **B** performs $B = g^b \bmod p$ and sends this to **A**. Computer **A** computes $s_a = B^a \bmod p$ and **B** computes $s_b = A^b \bmod p$. (Paar & Pelzl, 2010, pp. 206-208)

It can be proven that $s_a \equiv s_b$ as $(g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p$.

A third computer, **E**, intercepting the transmission would only have A, B, g and p which is not enough data to calculate s, providing that p is sufficiently large, because of the *discrete logarithm problem*. (Adrian, et al., 2015)

As the values are large, finding g^a is a time-consuming operation, as is finding the $\bmod p$ of that number. However, when these two operations are combined (Modular Exponentiation), the operation can be performed significantly more efficiently. In Java, this is done using `bnlib`, which performs the algorithm using Chinese Remainder Theorem and the Montgomery Reduction. (Zimmermann, n.d.)

The Java Programming Language

Java is a general-purpose programming language first released in 1995. A program written in the Java language can be run on almost any device without modification (Sun Microsystems, Inc., 1996), a design principle called *Write Once, Run Anywhere*. The program can be compiled into a single, executable .jar file which can be run on any device with Java installed by just opening the file.

Artefact Objectives

The primary objectives for the artefact were:

To produce a working implementation of the Diffie-Hellman procedure, which must:

a) Allow two computers connected to the same network to perform the algorithm
b) Allow the user to select the computer they wish to perform the algorithm with
c) Perform the algorithm in a secure manner
d) Display the final security key to the user(s)
e) Work on all major operating systems including Windows, mac OS and Linux
f) Have a Graphical User Interface (GUI)
g) Perform the algorithm in a reasonable length of time

These requirements will be used as a measurement of success to evaluate the quality of the final artefact.

Initial Research

Primarily, research was focused on five key areas:

1) How Diffie-Hellman works
2) How to program it
3) How to communicate between two computers
4) How to optimise the program
5) How to create a user interface

These areas were selected due to their importance in developing the final artefact. To research these areas, a variety of sources were used. To research the Diffie-Hellman algorithm itself, the most valuable resource was the original research paper for the algorithm, *Secure Communications Over Insecure Channels*. This clearly describes the procedure of the algorithm, and necessary considerations for the security of the procedure. More recent resources, including a cryptography textbook and subsequent papers on Diffie-Hellman were used to research development of the algorithm since its initial discovery.

Programming Language

One of the first research decisions was the programming language. The two primary considerations were Python and Java. The primary advantage of Python is that it is a simple and easy to write programming language with good specialist libraries (NumPy for example) to handle the large mathematical calculations that need to be performed. However, it is difficult to distribute a python program, especially if it consists of multiple files. Furthermore, the NumPy library is not included in Python's standard libraries, so it would have to be installed by the end user. The primary advantage of Java is that it compiles the final program into a single output file, including all dependencies, which will run on any computer with Java installed. Additionally, Java contains a standard `BigInteger` library which can efficiently perform the modular exponentiation operations that the program requires.

Evaluation of Success

Upon completion of the artefact, the objectives of the artefact were used to evaluate the completeness and success of the artefact.

a) Allow two computers connected to the same network to perform the algorithm

Providing, as the requirement states, that the computers are on the same network and there are no non-standard restrictions on the network, the program can connect to any other computer on the same network running the program. However, in some environments with restricted networks (schools or workplaces for example), administrator approval is required for the program to be able to connect to other computers on the same network. Unfortunately, there is no way to bypass this whilst maintaining the Peer-to-Peer nature of the program.

b) Allow the user to select the computer they wish to perform the algorithm with

This requirement has been partially met. The user can connect to another computer on the same network, but they do not *choose* the other computer as the requirement states. Instead, they enter the local IP Address of the other computer. Ideally the algorithm would provide a list of computers that have been detected to be running the program. However, this would require amendments to the protocol that the program uses to communicate.

c) Perform the algorithm in a secure manner

The algorithm uses realistic levels of security for a production requirement. The g and p values are a set of values intended for use in network cryptography, which are shown in *Appendix I: Numerical Constants*. Additionally, the 2048-bit key is significantly longer than the highest key length that can be reliably cracked. (Adrian, et al., 2015). The private keys are held in memory only and are discarded once the algorithm has finished.

d) Display the final security key to the user(s)

Upon the algorithm's completion, both computers display the key as shown in Figure 1 below. A *Copy* button is provided to copy the number to the user's clipboard. The key is in a format that can be used for most cryptographic algorithms, but it would also be useful to display it in hexadecimal.

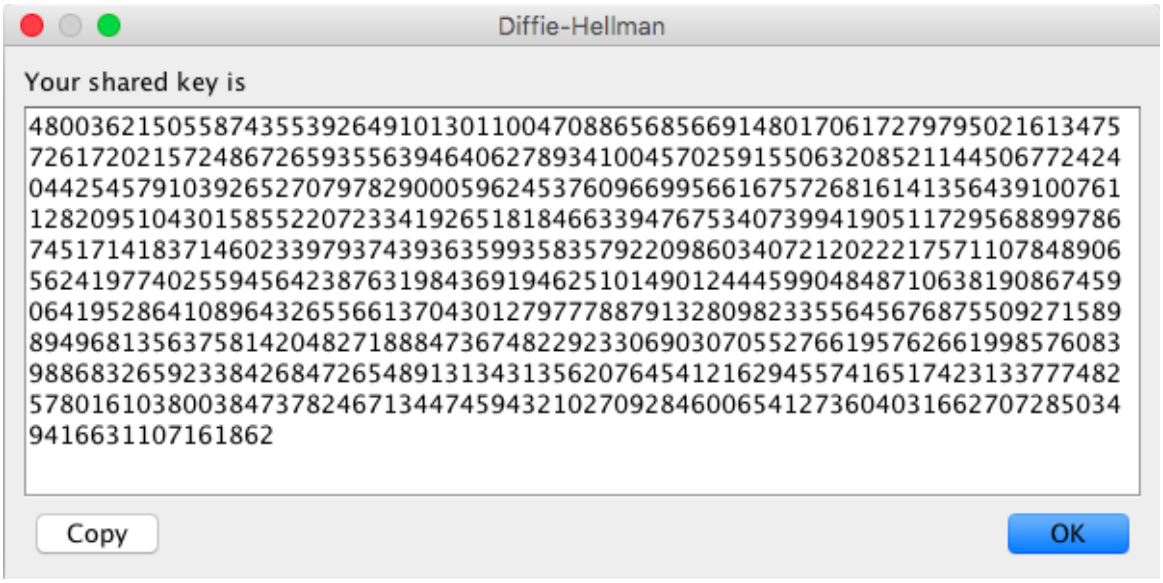


Figure 1 The result of the program

e) Work on all major operating systems including Windows, mac OS and Linux

Due to the Java programming language's *Write Once, Run Anywhere* design philosophy, the compiled `.jar` file can be run on any operating system with Java installed. There are some minor differences, for example the Server and Client buttons on the main screen's position are switched[1], but the program will still work. Compatibility was tested on a Windows 10 PC, a mac OS laptop and a Linux Virtual Machine, and in all cases the application worked correctly.

f) Have a Graphical User Interface (GUI)

The program has a GUI, through which all functions of the program can be performed. There are some minor issues with the interface, however these do not affect the operation of the program. Primarily, the "Working…" screen does not display properly on the computer acting as Client and the program appears unresponsive. This happens because the algorithm takes some time to run, and the operating system believes that the program has stopped responding. Resolving this issue would require an extensive rewrite of the project to include *threading* – the technique of running the algorithm and user interface in parallel on separate CPU threads. Secondly, the program does not prompt the user for the port to run the server on. This means that if the default port (2020) is in use, the program will be unable to start. Ideally the user should be prompted to enter a port, but a user-editable configuration file would also be satisfactory.

g) Perform the algorithm in a reasonable length of time

To test the length of time the application takes, it was run through an automated timing program on a Windows computer. It generally took approximately 3 seconds to compute the key. Whilst this is relatively quick, it is still too slow for some usages. There are some methods that can be used to decrease this time. Primarily, the computation parts of the algorithm could be rewritten in a lower-level programming language (for example C++), with the user interface code remaining in Java. Another possibility would be to write customised mathematical procedures specifically for the procedures in the algorithm. The fastest implementation of the algorithm takes approximately 400 milliseconds (Bernstein, 2006) to execute. However, this implementation does not strictly follow the algorithm to save time and relies on heavily optimised code for its mathematical operations, which must be rewritten for every type of

[1] This is because mac OS and Windows place the Primary and Secondary buttons in different places. This could be fixed in future versions of the program.

processor the software needs to be deployed on. For this reason, it is infeasible to use the techniques of this study to optimise the artefact.

Appendix I: Numerical Constants

The program uses the 2048-bit MODP Group for the Internet Key Exchange Standard (Kojo & Kivinen, 2003).

$g = 2$

$p = (2^{2048} - 2^{1984} - 1 + 2^{64}) \times (2^{1918\,\pi} + 124476)$

Which is equal to:

```
32317006071311007300338913926423828248817941241140239112842009751400
74170663435422261968941736356934711790173790970419175460587320919502
88537589861856221532121754125149017745202702357960782362488842461894
77587641105928646099411723245426622522193230540919037680524235519125
67971587011700105805587765103886184728025797605490356973256152616708
13393617995413364765591603683178967290731783845896806396719009772021
94168647225871031411336429319536193471636533209717077448227988588565
36920864529663607725026895550592836275112117409697299806841055435958
48665832916421362182310789909994486524682624169720359118525070453610
90559
```

The value of p is considered to be secure due to its usage of 'nothing-up-my-sleeve numbers'. It is possible for certain prime values to be weaker than others (Paar & Pelzl, 2010, pp. 72-75) but defining the number as an equation in terms of π, which cannot be changed, it shows that the prime has not been specifically chosen due to cryptographic weakness.

Bibliography

Adrian, D., Bhargavan, K., Durumeric, Z. & Gaudry, P., 2015. *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice.* New York, ACM, pp. 5-17.

Bernstein, D. J., 2006. Curve25519: New Diffie-Hellman Speed Records. In: M. Yung, Y. Dodis, A. Kiayias & T. Malkin, eds. *Public Key Cryptography - PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings.* Berlin, Heidelberg: Springer, pp. 207-228.

Corrigan-Gibbs, H., 2014. Keeping Secrets. *Stanford alumni*, November/December.

Diffie, W. & Hellman, M., 1976. NOVEMBER New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6).

Hellman, M. E., Diffie, B. W. & Merkle, R. C., 1977. *Cryptographic apparatus and method.* United States of America, Patent No. US4200770 A.

Kojo, M. & Kivinen, T., 2003. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE).* s.l.:RFC Editor.

Lee, Y.-R., Lee, H.-S. & Lee, H.-K., 2004. Multi-party authenticated key agreement protocols from multi-linear forms. *Applied Mathematics and Computation,* , 159(2), pp. 317-331.

Merkle, R. C., 1978. Secure Communications Over Insecure Channels. *Communications of the ACM*, 21(4).

Oracle, n.d. *Java Platform Standard Edition 8 Documentation.* [Online] Available at: https://docs.oracle.com/javase/8/docs/ [Accessed 24 November 2017].

Paar, C. & Pelzl, J., 2010. *Understanding Cryptography*. 2nd corrected printing ed. Berlin: Springer.

Sun Microsystems, Inc., 1996. *The Java Language Environment.* [Online] Available at: http://www.oracle.com/technetwork/java/intro-141325.html [Accessed 24 November 2017].

Zimmermann, P., n.d. *bnlib: Extended Precision Integer Math Library.* [Online] Available at: https://philzimmermann.com/EN/bnlib/index.html [Accessed 30 November 2017].

Appendix A: Presentation

Freddy Heppell

Securing Communications with the Diffie-Hellman Key Exchange

The Key Exchange Problem

If a cypher uses the same key to encrypt and decrypt a message, how can this key be sent securely to the recipient?

Send the key via a trusted courier
- Used for thousands of years, but insecure

Use an algorithm to generate the key
- First developed in 1976

Use an algorithm that uses two different keys
- First developed in 1977

Research

1. How Diffie-Hellman works
2. How to program it
3. How to communicate between two computers
4. How to optimise the program
5. How to create a user interface

What is the Diffie-Hellman key exchange?

Part I: Setup

1. Choose a prime p, say 29.
2. Choose a base g where $2 \leq g \leq (p-2)$, say 2.
3. Publish p and g.

Part II: Key Exchange

Alice	Bob
1. Choose a where $2 \leq a \leq (p-2)$ randomly, say 5.	1. Choose b where $2 \leq a \leq (p-2)$ randomly , say 12.
2. Compute $A = g^a \bmod p = 3$	2. Compute $B = g^b \bmod p = 7$
3. Exchange A and B publicly	
4. Compute $s = B^a \bmod p = 16$	4. Compute $s = A^b \bmod p = 16$

A shared value of s has been computed.

Research

- Original Research
  - *"New Directions in Cryptography" (1976)*
  - *Patent Filing (1980)*
- Modern Textbook
  - *"Understanding Cryptography" (2010)*
- Subsequent Papers
  - *"Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice" (2015)*
- Programming Research
  - *Language Documentation*
  - *IETF RFC 3538*

Artefact Aims

- To produce an implementation of the Diffie-Hellman algorithm
- It must:
  - Allow the user to select the computer they wish to perform the algorithm with
  - Allow two computers connected via a network to perform the algorithm and generate a shared key
  - Work on major operating systems, including Windows, mac OS and Linux
  - Have a graphical user interface (GUI) so it can be set up easily

Programming Language

Programming Language

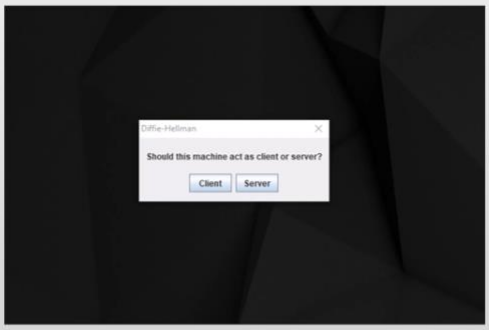
- Cross-platform
- Single file
- BigInteger library
- Swing interface framework

Planning

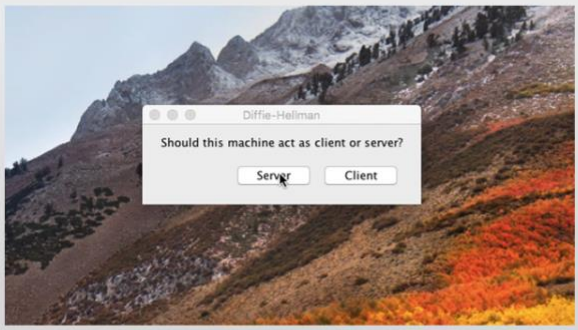
- Set artefact requirements
- Gantt chart for scheduling

Task	August				September				October				November				December			
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Topics Research																				
Decide Topic																				
D-H Research																				
Artifact Decisions																				
Create Artifact																				
Review Artifact																				
Writing Review																				
Prepare Presentation																				

Demonstration



Windows 10 PC



macOS Laptop

This slide contains videos demonstrating the artefact.

Artefact Code

```
// Send connection message
ConnectedMessage connectedMessage = new ConnectedMessage();
out.writeUTF(connectedMessage.toJSON());
System.out.println("Sent connection notice. Awaiting value receipt.");

// Wait for server to send A
InputStream inFromServer = client.getInputStream();
String receivedValueJSON = new DataInputStream(inFromServer).readUTF();

System.out.println(receivedValueJSON);

// Parse A
Map<String, String> receivedValue = Parser.parse(receivedValueJSON);
BigInteger A = new BigInteger(receivedValue.get("exchange_value"));

System.out.println(String.format("A=%d", A));

// Calculate B
BigInteger b = GenerateKey.generate();
BigInteger B = Settings.g.modPow(b, Settings.p);

System.out.println(String.format("b=%d\nB=%d", b, B));

// Send B to server
ExchangeValueMessage exchangeValueMessage = new ExchangeValueMessage(B);
String exchangeValueMessageJson = exchangeValueMessage.toJSON();

out.writeUTF(exchangeValueMessageJson);

// Calculate k
BigInteger k = A.modPow(b, Settings.p);
```

```
public static BigInteger generate() {
    SecureRandom random = new SecureRandom();

    int n = Settings.key_bound;

    SecureRandom r = new SecureRandom();
    byte[] b = new byte[n];
    r.nextBytes(b);

    return new BigInteger(b).abs();
}
```

```
ServerSocket serverSocket = new ServerSocket(port);

InetAddress ipAddress = InetAddress.getLocalHost();
String progressString = String.format("Awaiting connection on %s port %s", ipAddress.getHostAddress(), Integer.toString(port));
ProgressDialog awaitingConnection = new ProgressDialog(progressString);
Socket server = serverSocket.accept();
```

Artefact Code

$g = 2$

$p = (2^{2048} - 2^{1984} - 1 + 2^{64}) \times (2^{1918\pi} + 124476)$

32317006071311007300338913926423828248817941241140239112842009751400741706634354222619689417363569347117901737909704191754605873209195028853758986185622153212175412514901774520270235796078236248884246189477587641105928646099411723245426622522193230540919037680524235519125679715870117001058055877651038861847280257976054903569732561526167081339361799541336476559160368317896729073178384589680639671900977202194168647225871031411336429319536193471636533209717077448227988588565369208645296636077250268955505928362751121174096972998068410554359584866583291642136218231078990999448652468262416972035911852507045361090559

Reflection

- Researching algorithm
- Learning the Java programming language
- Creating the artefact

Evaluations

Problems

- Unresponsive windows
- Can be quite slow

Improvements

- Auto-detection of other computers on the same network
- Change from Client/Server architecture
- Lower level programming language

Questions?

Image Attributions:
Padlock By AJ Ashton (http://www.openclipart.org/detail/17931) [CC0]
Crypto key By MesserWoland CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)]
Python logo © Python Software Foundation
Java Logo © Oracle