Encryption and Cryptography

Bethany Waldmann

Follow this and additional works at: https://mosaic.messiah.edu/honors

Part of the Science and Technology Studies Commons

Recommended Citation
https://mosaic.messiah.edu/honors/94

Sharpening Intellect | Deepening Christian Faith | Inspiring Action

Messiah College is a Christian college of the liberal and applied arts and sciences. Our mission is to educate men and women toward maturity of intellect, character and Christian faith in preparation for lives of service, leadership and reconciliation in church and society.
1 Introduction

Information security\(^1\) is an on-going need in this digital age\(^2\). This security is absolutely necessary in applications where financial or identification information is transferred, but is also important in other applications where messages need to be protected. Cryptography is the art and science of keeping messages secure\(^6\), and is an important part of information security.

The purpose of this two-semester project is to study encryption and cryptographic techniques. Research was conducted on the history and direction of cryptography as well. Finally, to gain better understanding and put knowledge to use, a cryptographic algorithm was coded and implemented in an application.

2 Research

The first semester of work consisted of research. The bulk of the research is from Bruce Schneier’s *Applied Cryptography*[6], the only exceptions being elliptic curve cryptography and quantum computing: elliptic curve cryptography research was done with Behrouz Forouzan’s *Introduction to Cryptography and Network Security*[3]; quantum computing research was done with Isaac Chuang’s *Quantum Information: Joining the Foundations of Physics and Computer Science*[1].

\(^{1}\)See Appendix A for a definition.
2.1 Building Blocks

Cryptography is the art and science of keeping messages secure, but there are three additional popular goals of cryptography: authentication, so the receiver is satisfied that the sender did actually send the message; integrity, so the receiver is satisfied that the message could not have been modified in transit; and nonrepudiation, so that the sender cannot deny having sent a message.

It is necessary to understand some concepts before diving into specific cryptographic techniques. We cover terminology, security, protocols, one-way and hash functions, and random sequence generators.

2.1.1 Security

Security of a cryptographic algorithm can be achieved in two ways: through restriction of the algorithm, and through the use of keys. In the first, restriction of the algorithm, the details of the algorithm are not disclosed or made public. This provides security because to break the encryption, one would have to figure out how it works. With so many possible ways to do encryption, this may seem like a good solution, but in fact it is not a good idea because the algorithm could be leaked or stolen. This is especially dangerous because there may not be any evidence that the algorithm has been compromised.

The widely-accepted solution to ensure security is the use of keys. A key is a large number used as an input to a cryptographic algorithm. When different keys are used with the same plaintext in a given algorithm, the ciphertexts produced should not be the same.

2.1.2 Protocols

A protocol is a series of steps involving two or more people which is used to accomplish a task. There are three main types of protocols: arbitrated, adjudicated, and self-enforcing. In an arbitrated protocol, parties communicate through an arbitrator, or trusted third party; it is set up so that no party can cheat any other party. An adjudicated protocol is set up so that
parties hopefully don’t cheat each other, but in the case of a dispute an adjudicator is used. A self-enforcing protocol is one in which there cannot be any disputes; an arbitrator is not used.

### 2.1.3 One-Way and Hash Functions

A one-way function is a function that is easy to compute, but hard to reverse. Trapdoor one-way functions are one-way functions that are easy to reverse if the trick is known. Both these types of functions are useful in cryptography; the former in hash functions and the latter in public-key algorithms\(^2\).

A hash function is a special kind of one-way function that inputs a variable-length string and outputs a fixed-length string; for example, a hash function could convert any string to a string of length 64. Good hash functions are collision-free, meaning it’s unlikely that two inputs have the same hashed output. This kind of function is useful for keeping text secret, but being able to confirm later than some given text is either the same as or different than the hashed text; this is accomplished by comparing the hashed text with the hash of the input text.

### 2.2 Random Sequence Generators

Random sequence generators are useful for generating random numbers, and are especially needed to generate keys used in cryptography applications. The random number generators built into a number of programming languages are generally not secure enough for cryptography; this is because they use pseudo-random sequence generators, which look random but are predictable. When a real random sequence generator is used, the same input to the generator will result in different and, most importantly, unrelated sequences.

There are a lot of ways to achieve a truly random sequence, including RAND tables, random noise, and keyboard latency. Although the lower bits of a computer’s clock are fine for a few random bits, they are not sufficient for random sequences because of their regularity.

\(^2\)Public-key algorithms are discussed in 2.4.2.
Table 1: Cryptographic Attacks

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciphertext-only</td>
<td>Analyst has the ciphertext of several messages.</td>
</tr>
<tr>
<td>Known-plaintext</td>
<td>Analyst has ciphertext and plaintext of some messages.</td>
</tr>
<tr>
<td>Chosen-plaintext</td>
<td>Analyst has the ciphertext and plaintext of chosen messages.</td>
</tr>
<tr>
<td>Adaptive-chosen-plaintext</td>
<td>Like chosen-plaintext, but the analyst can modify their choice based on previous results.</td>
</tr>
<tr>
<td>Chosen-ciphertext</td>
<td>Analyst chooses the ciphertexts to be decrypted.</td>
</tr>
<tr>
<td>Chosen-key</td>
<td>Analyst chooses the ciphertexts to be decrypted.</td>
</tr>
<tr>
<td>Rubber-hose</td>
<td>Analyst threatens, blackmails, or tortures someone to make them give up the key.</td>
</tr>
</tbody>
</table>

Even when input is truly random, the instruments measuring the randomness may introduce bias\(^3\). One solution to bias is to exclusive or\(^4\) (XOR) a number of bits together; this requires an assessment of the amount of bias that is acceptable, because the number of bits XORed together determines how much bias is SOMETHING. A second solution is to look at the bits in pairs and output a bit based on their relationship to each other. Unfortunately, if the bits are correlated,\(^5\) this will produce even more bias. The best solution is to distill randomness by hashing multiple random sources together.

### 2.3 Attacks

In addition to the general types of attacks shown in Table 1, there are three specific types of attacks: birthday, meet-in-the-middle, and man-in-the-middle.

The birthday paradox illustrates that it is easier to find two things that are the same than find a match for a specific thing; although there only need to be 23 people in a room for there to be a better than 50% chance that two of them share the same birthday, there would need to be 253 people in the room for a better than 50% chance that a specific person shared the same birthday with someone else. In cryptography, a birthday attack can be performed on a hash algorithm; the attacker tries to find two plaintexts that hash to the same value, rather

\(^3\)See Appendix A for a definition.
\(^4\)See Appendix A for a definition.
\(^5\)See Appendix A for a definition.
than trying to find a second plaintext that hashes to the same value as a given plaintext.

Double encryption is when the output of encryption with one key is used as input with a second key. A meet-in-the-middle attack can be performed on double-encrypted block algorithms\(^6\) by encrypting from one end and decrypting from the other. Using this attack can roughly halve the number of encryptions required to break an algorithm.

Man-in-the-middle attacks occur when the attacker intercepts a key exchange and substitutes their own input for the sender’s. When the receiver sends a message back encrypted with the key, the attacker can decrypt it with the substituted key and encrypt it with the sender’s original key before sending it on to the original sender. This is a very dangerous attack because the involved parties do not know that their security has been compromised.

## 2.4 Algorithms

There are two general types of cryptographic algorithms: symmetric and public-key.

### 2.4.1 Symmetric Algorithms

Symmetric algorithms use the same key for encryption and decryption; the key must be agreed upon before communication. There are two different types of symmetric algorithms: stream ciphers and block ciphers.

Stream ciphers encrypt bit by bit using a keystream generator, which is a stream of random bits. The security of the encryption depends on the strength of the keystream generator, and there are many possible implementations. To get a bit of ciphertext from a stream cipher, XOR a plaintext bit with a bit from the keystream generator. In self-synchronizing stream ciphers, each keystream bit is dependent on a number of previous ciphertext bits and requires each message to begin with some random bits so that nothing is lost. While self-synchronizing stream ciphers are handy, they are vulnerable to the playback attack in which a previously-sent message is sent again; this attack is particularly harmful in banking, since someone could have

\(^6\)Block algorithms are discussed in 2.4.1.
more and more money “deposited” into their account.

Block ciphers encrypt chunks (blocks) of ciphertext at a time. Blocks are generally 64 bits, but can be different sizes. Message lengths are not always a multiple of 64 bits, so padding is required. To pad a message, the short block is filled to the end with any regular pattern and a last block with the number of padding bytes is added to the end so they can be removed during decryption. There are two modes that block ciphers can use: Electronic Codebook Mode (ECB), in which blocks are encrypted and decrypted separately; and Cipher Block Chaining (CBC) mode, in which feedback is added by XORing the plaintext block with the previous ciphertext block before encryption. Block ciphers always encrypt to the same ciphertext, regardless of the mode used.

2.4.2 Public-Key Algorithms

Public-key algorithms are based on hard problems. The most popular hard problems used in public-key cryptography are factoring, discrete logarithms, and elliptic curves. Rather than using the same key for encryption and decryption, or even two keys that can generate each other, public-key algorithms use two keys which are related but cannot generate each other. One of the keys is public and distributed freely for encryption; the other is private and kept secret for decryption. While the public key is usually used for encryption and the private key for decryption, the algorithm will work the opposite way; this is especially useful in digital signing.

To send a message, all one has to do is encrypt their message with the receiver’s public key. Once encrypted, only the receiver will be able to decrypt the message with their private key. To prove their authorship in a digital signature scheme, the sender can encrypt the message with their private key before using the receiver’s public key for encryption; when the receiver gets the message, they will decrypt the message with their private key and then with the sender’s public key.
2.5 Digital Signatures

Digital signatures are used to prove authorship of a message. Digital signature schemes must prove the following to the receiver: that the sender deliberately signed the document; that only the sender could have signed it; that a third party cannot use the signature on a different document; that the document is unalterable once signed; and that the sender cannot deny having signed the document. Digital signature schemes are generally implemented with some combination of hashing and encryption.

2.6 Quantum Computing

The encryption algorithms discussed above are used because it is thought that breaking one requires brute force\(^7\). Strong algorithms have been designed so that using brute force is infeasible. These algorithms are based on a thesis that problems are easy or hard regardless of the physical world, which is false in quantum physics.

In classic physics, an atom’s electrons can be in their ground state or in an excited state. Perhaps the example of this that is best known to the general public is that of neon lights. When electricity is run through a neon gas, its electrons move to an excited state; when it drops down from an excited state, orange light is emitted. At any given time the electron is either in its ground state or an excited state – they are mutually exclusive. However, in quantum physics, atoms can be in a superposition state. In this superposition state, the atom is in both its ground state and an excited state simultaneously, but any measurement of its state will show that it is in a discrete state (ground or excited) rather than somewhere inbetween. Furthermore, there is entanglement in which two atoms’ states are linked together regardless of how far apart they are. When entangled atoms are in a superposition state, measurement will find them both to be either grounded or excited.

An application of quantum physics could build a quantum computer in which mathematical expressions are evaluated simultaneously for all possible inputs. Since encryption is based on

\(^7\)See Appendix A for a definition.
mathematics, brute force would no longer be time-consuming. Scientists have not been able to create a quantum state in a large enough object to constitute a quantum computer, but this March they were able to create a quantum state in an object billions of times larger than was previously found[5]. While it’s uncertain whether a quantum computer will ever be built, it seems likely. When that happens, a shift in the methods of cryptography will occur.

3 Project

The second semester consisted of implementing one of the encryption/decryption schemes for an application. I chose to use the Data Encryption Standard (DES) algorithm to encrypt and decrypt text files.

3.1 DES

DES is a 64-bit block algorithm with a 56-bit key. The key is input as a 64-bit number, but every eighth bit is an odd parity⁸ check and discarded before the key is used.

First the block is permuted⁹ with an initial permutation. Then the key is reduced to 56 bits with a compression permutation that removes the parity bits. The block is put through 16 rounds before being joined and permuted a final time.

A DES round consists of the operations pictured in Figure 3.1. First, a copy of the right half of the block is saved and the key is divided in half and shifted left circularly¹⁰ once or twice depending on the round. After the key is joined back together and a copy saved for the next round, it is compressed to 48 bits with another compression permutation. Next the right half of the block is expanded to 48 bits with an expansion permutation and XORRed with the reduced key. The result is fed into eight S-boxes¹¹, reducing it back down to 32 bits before it is permuted in the P-box, XORRed with the left half, and saved as the new right half. Finally, the

---
⁸See Appendix A for a definition.
⁹See Appendix A for a definition.
¹⁰See Appendix A for a definition.
¹¹S-boxes are described in the next paragraph.
left half is replaced with the copy of the right half made at the beginning of the round. These new halves and the copy of the key before it was compressed are used in the next round. After the last round, the left and right halves are switched, joined together, and permuted one last time. An equivalent algorithm computes the round keys before entering the rounds, which is more efficient when more than one block is being encrypted.

An S-box is a substitution table with 4 rows and 16 columns in which each cell is a 4-bit value. A cell in the table is selected based on a 6-bit input which determines the row and column; with an input of $b_0b_1b_2b_3b_4b_5$ the row is $b_0b_5$ and the column is $b_1b_2b_3b_4$, both interpreted in binary\textsuperscript{12}.

All of the permutations and S-boxes were carefully picked by DES’s creators so that the algorithm would be circular – that is, with a small adjustment, the algorithm can be used for decryption. The small adjustment is to use the round keys in the opposite order, which can be done in two ways: if the round keys are generated during the round, the shifting needs to be reversed to the right using a different set to determine how many bits to shift in each round;

\textsuperscript{12}See Appendix A for a definition.
if the keys are generated beforehand, they can be reversed before use or used backward inside the rounds.

### 3.2 Implementation

My original idea was to program DES in C++ for a 16-bit HCS12 microprocessor borrowed from the Engineering department, which was cleared by Department Chair Dr. Randall Fish. After implementing both encryption and decryption, I would use it in a specific application of my choice. However, I ran into a few problems along the way which changed the project and set my work back considerably. The end product is a standard C++ program that can encrypt and decrypt text files using DES.

#### 3.2.1 Problems and Solutions

**Compiling** The first problem occurred when I was about halfway through the algorithm: the program would not compile. After looking into the error code, I discovered that the software I was using\textsuperscript{13} to compile code for and interface with the microprocessor limits the amount of code that can be compiled in C++.

That left me with the following options: 1) purchase an edition of the software that would compile a larger amount of C++ code, which would cost $1495 with a student discount; 2) switch the language to C, which the free software will compile up to 32 times more of compared to C++; 3) switch the language to Assembly, which the free software will compile an unlimited amount of; or 4) stop using the microprocessor altogether and switch to a traditional C++ application. Option (1) was not feasible for this small-scale project. Option (2) was feasible, but a quick attempt to change the file extension to C did not compile and given my lack of knowledge of C, I did not want to spend more time porting the code. Option (3) was more accessible given that I already knew Assembly language, but the compilations of operating on 64-bit numbers on a 16-bit processor would require even more time than switching to C.

\textsuperscript{13}Freescale CodeWarrior’s Special Edition, which is free of charge, was used.
Thus Option (4) was chosen for the simplicity of the transition and the small amount of time it would take to make the transition.

**Indexing**  The second problem I encountered was indexing (though I did not know it at the time). After writing and testing code to do a permutation, I moved on to tackle something else. However, when I attempted to use the method in the algorithm with the DES permutations listed in my source[6], it did not work correctly. However, I noticed that one half of the bits was correct but in the wrong half; on a smaller scale for example, if I was expecting 01011010 I might get 11101010.

I spent many hours pouring over the code and testing the individual sections that perform permutation; my professor and I even pair-programmed\(^{14}\) a program in Java to assist the process, in which we coded a permutation algorithm that worked correctly. Finally, after approximately 10 hours of bewilderment and frustration, I realized that the permutations I got from my source were 1-based indexing\(^{15}\), whereas my permutation algorithm assumed 0-based indexing.

I had not immediately noticed this because C++ does not produce errors when an array is accessed with an index greater than its length; instead, it simply returns whatever is at the memory location outside it. Although I did not at any point in the process consciously decide that I was accessing the correct indices in the array because of a lack of error that was produced, it must have been in my mind subconsciously; indeed, until my epiphany, the idea that the wrong indices were be accessed did not cross my mind. Needless to say, once the reason for this problem was discovered it was an easy fix.

**Orientation**  The final problem I had was bit orientation\(^{16}\). My source did not outrightly say whether the DES algorithm required the least significant bit to be on the left or on the right, and I assumed without consideration that it was on the left because of the standard

\(^{14}\)See Appendix A for a definition.
\(^{15}\)See Appendix A for a definition.
\(^{16}\)See Appendix A for a definition.
representation of binary.

After finishing the encryption algorithm its decryption counterpart, unit testing on all parts of the code convinced me that the algorithm would work properly. However, when I encrypted and then decrypted, I did not get what I started with; which meant that the encryption or decryption was wrong, and possibly both. Further testing was done to no avail before I decided that a second source on the algorithm was in order.

My second source, *Handbook of Applied Cryptography* [4] by Alfred Menezes, revealed exactly the same algorithm, with examples showing the least significant bits on the right. I continued to disregard bit orientation because I was convinced that the orientation did not have an impact since I was accessing and storing the bits backward. However, after more frustration, I realized that although I was accessing the bits backward, a few other things would need to be changed for it to produce a result in the opposite order of the apparently-traditional representation.

After making the necessary changes to invert the algorithm completely, the algorithm was still not producing the correct result. I checked the inverted solution with my professor, and he agreed that it seemed to be correct. Finally, after checking all the code again, I decided to switch the bit orientation and return the algorithm to its original state – and it worked! All in all, this was the most time-consuming problem I had as it took approximately 30 hours to troubleshoot and fix.

4 Future Work

There are a few possible expansions of this DES project. One expansion could be to increase the security of encryption by implementing triple-DES, which uses a 112-bit key. Another expansion could be to use the algorithm in a different application. A third expansion could be adding a hashing algorithm and implementing a digital signature scheme.

Looking outside DES, a public-key algorithm could be implemented. When complete, a
cryptosystem using the public-key algorithm for key exchange and DES for encryption could be created.

5 Conclusion

During this two-semester project, I learned a lot about cryptographic protocols, techniques, and their possible future through both my research and DES implementation. Though I had many frustrations, they taught me valuable lessons. Overall, I am glad that I did this project.

6 Acknowledgments

I would like to thank my project advisor, Professor D. Scott Weaver, for all the help and guidance he provided. I would also like to thank the Department of Mathematical Sciences for allowing me to do this project, and the Department of Engineering for their cooperation.
References


## A Definitions

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bias</td>
<td>The average is not what is statistically expected.</td>
</tr>
<tr>
<td>Binary</td>
<td>A representation of numbers in base 2; rather than using the digits 0–9 in the base 10 system used in the decimal system, only 0 and 1 are used in binary.</td>
</tr>
<tr>
<td>Brute force</td>
<td>A method that systematically checks answers according to some algorithm until a solution is found.</td>
</tr>
<tr>
<td>Circular shift</td>
<td>A shift operation that moves the bits in a wrapping fashion.</td>
</tr>
<tr>
<td>Correlation</td>
<td>A relationship between variables which is not statistically expected.</td>
</tr>
<tr>
<td>Exclusive or</td>
<td>A binary function on a bit that inputs multiple bits and outputs a single bit depending on whether the number of ones is even or odd.</td>
</tr>
<tr>
<td>Indexing</td>
<td>Determines what number counting starts at; 0-based indexing starts at 0, whereas 1-based indexing starts at 1.</td>
</tr>
<tr>
<td>Information security</td>
<td>The field of research that aims to protect information from malicious attackers while still allowing legitimate users to manipulate data freely.[2]</td>
</tr>
<tr>
<td>Orientation</td>
<td>Bit orientation, in the context of bit strings, determines whether the least significant bit is represented on the left or right side of the string.</td>
</tr>
<tr>
<td>Pair Programming</td>
<td>A programming method in which two people sit at a computer; one person controls the keyboard and mouse while the other provides feedback.</td>
</tr>
<tr>
<td>Parity</td>
<td>Used to check the number of bits that are set; odd parity produces a 1 when there are an odd number of ones, otherwise it produces a 0.</td>
</tr>
<tr>
<td>Permutation</td>
<td>Rearranging of objects, often numbers. A regular permutation uses exactly all objects exactly once, but there are expansion and compression permutations.</td>
</tr>
</tbody>
</table>