Cambridge Academy for Science and Technology

AQA Computer Science

Practical Computing Project

CRYPTOGRAPHY ONLINE

Author
J.P. Jacob Powell

Supervisor
B.C. Barry Cooper

March 21, 2018

Contents

Chapter 1

Analysis

1.1 What is Cryptography

Cryptography is idea of allowing 2 parties to communicate securely and not allowing an adversary to listen in on the communications. In a general sense it is the art of building secure protocols that allow us to communicate securely. This idea has been around for centuries and is constantly evolving with the increase in demand for technology in our everyday lives and the need make certain that everything that is in the public domain is secure.

1.1.1 Quick History

1.1.1.1 Ancient History

One of the first recorded instances of cryptography is from the Roman Empire when *Julius Caesar* created what is now known as the Caesar Cipher. This cipher works by shifting the letters of the alphabet a given number of times. According to *Gaius Suetonius Tranquillus*, Julius Caesar used a shift of +3 places to protect messages of military importance. Even though it is unknown how effective the cipher was at the time it can be assumed that it was reasonably secure. Not least because most of Caesars enemies would have been illiterate and others would have just presumed that the messages were written in an unknown foreign language.

At the time there was no recorded way to break these ciphers. The earliest records of attacks that could be used to break the cipher are from the 9^{th} century, with the work of *Al-Kindi* in the Arab world with the discovery of frequency analysis.

1.1.1.2 Recent History

WW1: One of the most notable events during WW1 would be the British interception and decryption of the Zimmermann Telegram. Room 40, the section in British admiralty most identified with cryptanalysis effort during WW1, heavily contributed to the decryption of the Zimmermann Telegram.

This decryption is often referred to as the most significant triumph in signals intelligence for Britain in the First World War, and one of the first times in which a single piece of signals intelligence directly influenced world events. In this case the United States of America joined the War on 6^{th} April 1917.

WW2: During World War 2, cryptography was used extensively with a plethora of ciphers systems fielded by the nations involved during the war. In addition to the advancements of cipher systems, the theoretical and practical aspects of cryptanalysis was much more advanced.

Probably the most important codebreaking event during WW2 was the breaking and decryption of the *Enigma* Cipher. The first 'full' break of the enigma cipher was in 1932 by Poland, with the techniques and insights were passed on to the British and French Allies just before the start of the War in 1939. These were improved significantly by the British efforts at the Bletchly Park research station during the War. The decryption of the Enigma Cipher allowed the allies to read important parts of the German Radio networks and it provided an invaluable source of military intelligence throughout the war. Intelligence from this source (including other high level sources, including the Fish Ciphers), was eventually called *Ultra*.

A similar break into an important Japanese cipher (PURPLE) by the US Army Signals Intelligence started before the US entered the War. The product of this effort was given the codename *MAGIC*, it was the highest security Japanese diplomatic cipher.

1.1.2 Modern Day Cryptography

1.1.2.1 Symmetric

Symmetric-Key Algorithms are algorithms for cryptography that use the same Key for encryption and decryption of the plaintext and ciphertext respectively. These *Keys* are a representation of a shared secret between 2 or more parties, that can be used to maintain a confidential link of communication. The main drawback of this system is that all the parties communicating need to know the secret key in order to communicate with each other.

There are 2 main types of Symmetric Cryptography Algorithms:

- Stream Ciphers

  Stream ciphers will go through each bit of the plaintext and encrypt it one at a time

- Block Ciphers

  Block Ciphers will divide the plaintext into groups of bits and will encrypt each of those *Blocks* at once, padding the plaintext if the length of the plaintext does not equally divide the block size of the cipher. Blocks of 64-bits were commonly used but now a days a block size of 128-bits is more common. The Advanced Encryption Standard (approved by NIST in 2001) and the GCM Block cipher both use block sizes of 128-bits.

1.1.2.2 Asymmetric

Public Key Cryptography or Asymmetric Cryptography is any cryptographic system that instead of using a single secret key, uses pairs of keys:

- Public Keys

  These could be known widely to the public domain

- Private Keys

  These are only known to the Owner

This accomplishes 2 things: Authentication, where the public key verifies that the holder of the paired private keys sent the message, and encryption, where only the paired private key holder can decrypt the message encrypted with the public key.

Public Key systems are often based on complex mathematical systems that currently have no reasonable solution to them. Some of these include certain integer factorization, discrete logarithm and elliptic curve problems. Unlike symmetric key algorithms, asymmetric algorithms do not require a secure channel of communication for the initial exchange of one or more of the secret keys between the parties.

Because of the computational complexity of asymmetric cryptography, it is usually only used for small blocks of data. This is typically the transfer of a symmetric encryption key or *Session Key*. This session key is the used to encrypt the rest of the potentially longer message. This symmetric encryption/decryption is based on simpler algorithms and is much faster.

In a public key signature system a person can combine their message with a private key to create a short digital signature on the message. Anyone with the corresponding public key can combine a message, a digital signature and the known public key to verify if the signature is valid or not. Therefore this provides authentication of a message provided that the owner of the private key keeps the private key secret.

These public key algorithms play fundamental roles in cryptosystems, applications and protocols. They are heavily involved in internet standards such as TLS (Transport Layer Security), S/MIME, PGP and GPG. Some public key algorithms provide key distribution and secrecy. Examples include the Diffie-Hellman Key Exchange. Some algorithms provide digital signatures. Examples include the Digital Signature Algorithm and some provide both, e.g. RSA.

Public Key Cryptography finds applications in the Information Security Discipline. Information Security (IS) is concerned with all aspects of protecting electronic information assets against security threats. Public Key cryptography is used as a method of assuring the confidentiality, authenticity and non-reputability of electronic communications.

1.1.2.3 Cryptanalysis

Cryptanalysis is the study of information systems working towards the goal of unveiling the hidden aspects of the system. This knowledge is then used

to break the information system and gain access to the contents of encrypted information.

In addition to mathematical analysis of cryptographic algorithms, cryptanalysis also involves looking at side-channel attacks. These attacks don't specifically look for weaknesses in the algorithm itself but for a weakness in the implementation of algorithm.

Considering the end goal has been the same, the specific ways in which we get there have evolved and developed drastically of the past few decades. We have gone from the pen-and-papers methods of the past, to the machines like the British bombes and colossus supercomputer at Bletchly part to the mathematically advanced computerized schemes of the preset. Most cryptanalysis of modern day algorithms involves solving complex pure mathematical problems, including the integer-factorization problem and the discrete logarithm problem.

1.1.2.4 Protocols

A security protocol is an abstract protocol that performs a security function and applies cryptographic methods, often grouped together as sequences of cryptographic primitives. The protocol defines how the algorithms should be used.

Cryptographic protocols are widely used for application-level data transport. A cryptographic protocol usually incorporates at least a few of the following features

- Key Agreement / Establishment
- Entity Authentication
- Symmetric Encryption and message material construction
- Secure Application-level data transport
- Non-repudiation methods
- Secret Sharing Methods
- Secure Multi-party computation

Example: Transport Layer Security An example of one of these algorithms could be the Transport Layer Security, or TLS, protocol. TLS is used to secure web (HTTP/HTTPS) connections. It has an entity authentication mechanism based on the X.509 system; a key setup phase, where a symmetric key is formed by employing public-key cryptography; and a secure application-level data transport system. These 3 functions have very important interconnections. Standard TLS does not however implement Non-repudiation methods.

There are other types of cryptographic protocols as well, and even the spelling has multiple meanings. Cryptographic Application Protocols, often

use one or more underlying key agreement methods, which are sometimes themselves referred to as cryptographic protocols. For example, TLS employs the Diffie-Hellman Key Exchange, which although is only a part of the TLS protocol, could be seen as a completely independent cryptographic protocol in itself for other applications.

1.1.3 Applications of Modern Cryptography in Everyday Life

Cryptography is present in nearly every single electronic device that has been produced in the past decade. This ranges from your new ePassport to your new mobile phone to your smart fridge. It is everywhere and because of this it has become even more important.

For the most part Cryptography provides security to a service that we use in out daily lives. Examples of these services could be:

- ATM Cash Withdraw
- File Storage
- Emails
- TV
- Text Messaging
- Web Browsing
- Payment Systems like PayPal

1.2 Algorithms Being Used

1.2.1 DES

1.2.1.1 What is DES?

DES, or the Data Encryption Standard, is a symmetric block cipher used for the encryption and decryption of electronic data. Even though it has now been proven that DES is insecure it was a major keystone in the development of modern cryptography.

Developed in the early 1970's by IBM, based of an earlier design by *Horst Feistal*, the algorithm was submitted to the National Bureau of Standards (NBS) following the agency's invitation to propose a candidate for the protection of unclassified, sensitive electronic government data. In 1976, after consultation with the National Security Agency (NSA), the NBS eventually selected a modified version of DES which was published as a Federal Information Processing Standard (FIPS) for the United States in 1977. It should be noted that this modified version helped protect against differential cryptanalysis but weakened it significantly against brute-force attacks, the attack that

would eventually make NIST (National Institute for Standards and Technology, the new NBS) ask for the newer Advanced Encryption Standard (AES).

1.2.1.2 Overview of DES

DES is a Block Cipher. It takes a fixed-length amount of plaintext bits and transforms it through a series of complicated operations which results in ciphertext bits. In the case of DES, its block-size is 64-bits, this means that you would split up your plaintext into 64-bit blocks and pass them through the algorithm one at a time. In the case that your plaintext does not fully divide into 64-bits you would use padding to increase the size of the plaintext until it fully divided into 64-bits. DES also uses a Key that heavily effects the transformation, so decryption can only be done if the user has the Key. DES uses a Key size of 64-bits. However it should be noted that 8 of these bits are used solely for parity checks so the effective Key size of DES is 56-bits. The Key is stored or transferred as 8 bytes with odd parity. According to *ANSI INCITS 92-1981*, section 3.5:

> *One bit in every byte of the KEY may be utilized for error detection in key generation, distribution and storage. Bits 8,16,..,64 are for use in ensuring that each byte is of odd parity.*

Like all Block Ciphers, DES on its own in not inherently secure, it must be used in a mode of operation to achieve this. FIPS-81 specifies several modes to be used with DES.

Decryption works the same way as encryption, but the keys are used in reverse order. This gives the advantage that you can use the same hardware/software for both encryption and decryption.

Overview of Internal Structure: DES uses 16 rounds, this means that the core blocks of the cipher are repeated 16 times on each block of plaintext. There is also an Initial Permutation and a Final Permutation, name IP and FP respectively. These have no significant cryptographic impact on the cipher but were added to allow the loading of blocks in and out of mid-1970s 8-bit based hardware.

Before the main rounds the block is divided into two 32-bit halves, being processes alternatively. This structure is known as a Feistel Network. This way of constructing a block cipher ensures that decryption and encryption are very similar processes, the only difference being that the subkeys are applied in reverse for decryption. Figure 1.1 shows this.

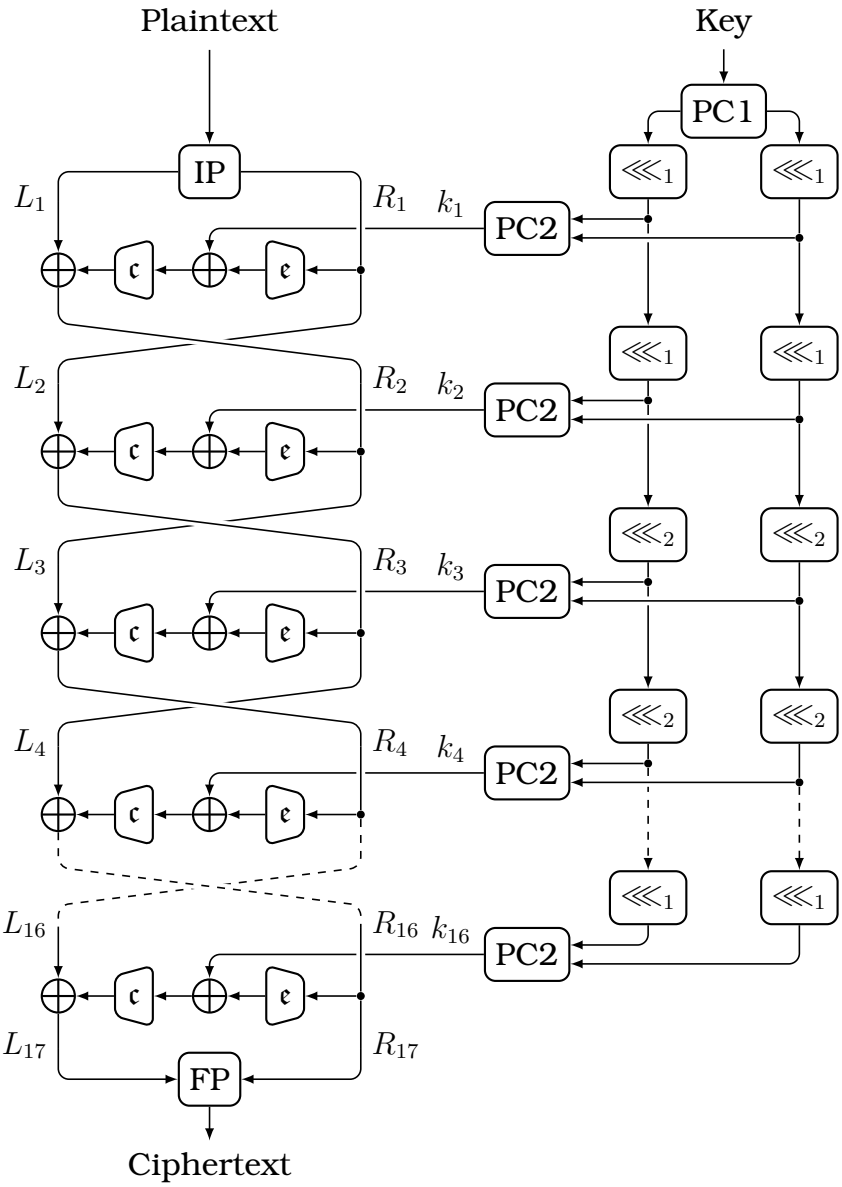

Figure 1.1: High Level Overview of the DES Block Cipher

1.2.2 AES

1.2.2.1 What is AES?

AES, or the Advanced Encryption Standard, is a specification for the encryption of electronic data established by the National Institute for Standards and Technology (NIST) in 2001. AES is a subset of the Rijndael Cipher, developed by 2 Belgium cryptographers. *Vincent Rijmen* and *Joan Daemen*, who submitted the cipher during the AES selection phase. Rijndael is a family of Block Ciphers which support multiple block sizes and key lengths.

For AES, NIST selected only 3 members of the Rijndael cipher. Each using

a block size of 128-bits and supports using key lengths of 128, 192, 256 bits.

AES was first adopted by the US Government and nowadays is used world-wide. It has become the defacto standard for securing electronic data. It has replaced the Data Encryption Standard which was first published in 1977. AES was first announced by NIST on *November 26, 2001* as a FIPS PUB 197 (FIPS 197). This announcement followed a five year standardization process on which fifteen competing designs were presented and evaluated, before the Rijndael Cipher was accepted as the most suitable.

AES became effective as a federal government standard on *May 26, 2002*, after its approval by the state of commerce. AES is included in ISO/IEC 18033-3 Standard. AES is available in multiple encryption packages and is the first(and only) cipher approved by the National Security Agency (NSA) for top secret information when used in an NSA cryptographic module.

1.2.2.2 Overview of AES

AES is based on what is known as a substitution-permutation network, and is very fast in both hardware and software. Unlike its predecessor DES, AES is not build on a Feistal Network. Since AES is a variant of the Rijndael Cipher using a Block size of 128-bits and supporting key sizes of 128, 192, 256 bits.

AES operates on a 4 x 4 column major order matrix of bytes, known as the state of the cipher. The calculations of this Cipher are performed in the Galios Field of 2^8, $GF(2^8)$, this is covered in detail in *Section 2.4.1*. The byte ordering in the matrix is shown in Figure 1.2.

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

Figure 1.2: AES Byte Ordering

The Key size used for the AES cipher specifies the number of rounds the used to convert the plaintext block into the ciphertext block.

- 128-bits : 10 Rounds
- 192-bits : 12 Rounds
- 256-bits : 14 Rounds

Each round consists of several processing steps, each containing 4 steps including one that depends on the key supplied to the algorithm. A set of

reverse rounds are applied to transform the ciphertext back into the plaintext using the same key. A High Level description of the algorithm is given below in Figure 1.3:

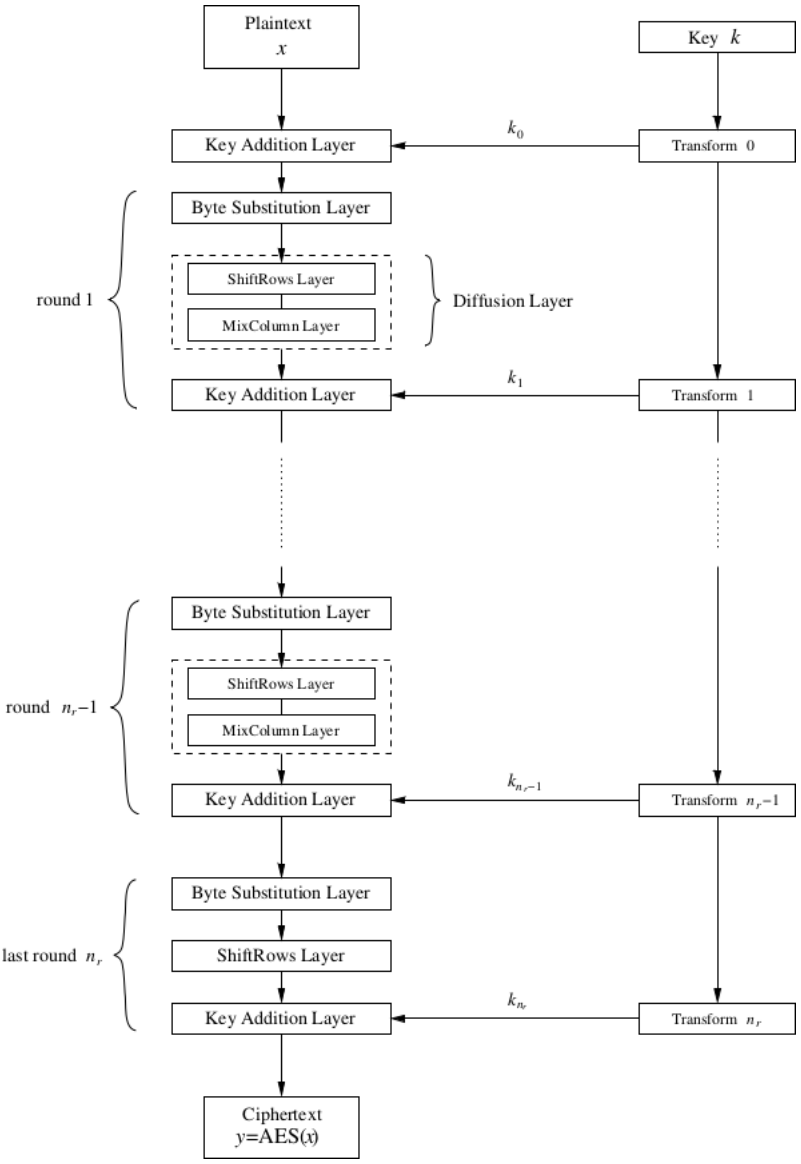

Figure 1.3: AES High Level Overview

1.2.3 SHA-3

1.2.3.1 What is SHA-3

SHA-3, commonly known as the Secure Hashing Algorithm-3, is the most up to date member of the Secure Hashing Algorithm family. It was released by NIST on *August 5, 2015*. Although it is a member of the same group

hash function standards, SHA-3's internal structure is quite different from the MD5-like structures of SHA-1 and SHA-2.

SHA-3 is a subset of a broader group of cryptographic primitive Keccak, designed by *Guido Bertoni*, *Joan Daemon*, *Micheal Peeters* and *Gilles Van Assche*, building upon their previous RadioGatun cryptographic primitive. Keccaks authors have proposed further additional uses of the hash function, which are not yet standardized by NIST, including a stream cipher, an authenticated encryption system, a 'tree' hashing scheme for faster hashing on certain architectures and AEAD ciphers Keyak and Ketje.

Keccak is based on something known as sponge construction. Sponge construction is based on a wide random function or random permutation, and allows inputting of any size of data and outputting any size of data, while acting as a psuedrandom function with regards to all previous inputs. This allows SHA-3 to have great flexibility.

NIST does not have any plans to remove SHA-2 from the revised Secure Hash Standard. The purpose of SHA-3 is that it can be directly substituted for SHA-2 in current application if necessary, and to significantly improve the robustness of NIST's overall hash algorithm toolkit.

1.2.3.2 Overview of SHA-3

I NEED TO LEARN THE STRUCTURE OF SHA-3

1.3 Why is this important?

1.3.1 The Importance of Cryptography in the Modern day

In the modern era, the amount of reliance we have on technology has exponentially increased over the past few decades. With this dramatic increase, we have started using this new technology to complete tasks that use very important and critical information about us, like online banking. This has opened a new opportunity for criminals to gain access to your critical accounts and now, rather than having to rob a bank, steal your credit cards, mug you, they only need to be in the same coffee shop connected to the same pubic Wifi as you. This is a very bad situation.

Cryptography plays a key role in this as it is the only thing that stops attackers from being able to access all of our critical information. As long our data is encrypted securely, it doesn't matter whether or not they can see what we are doing by being connected to the same public Wifi, all they will see is jiberish that makes no sense. And as long as the algorithms we use are secure and we make sure they don't get access to our private cryptographic keys they will never be able to access our critical data.

This is why Cryptography is very very important in the Modern Day. Cryptography is the only defense we have against attackers who have access to our data. The counter argument could be made that as long as they never get

access to our data we will be fine. But, recent cyber attacks against corporate companies have proven that even the most secure facilities are not 100% secure. So preventing the attackers from getting the data in the first place is important, but it is equally important to make sure that when they get the data, that data is useless to them.

1.3.2 Analysis of Current Solutions

Now we know that cryptography is possible one of the most important concepts in our digital age we need to make sure that it stays that way. This means we need to educate and train the next generation of cryptographers so that in 100 years time, we don't run out of people that can develop, test, maintain the next wave of cryptographic algorithms. So, what are the current ways that an individual can learn the ways of the cryptographers.

1.3.2.1 1. Academia

The most traditional way to break into a given industry, you enroll at a institution that allows you to study cryptography. You will learn everything, from the basics of cryptography all the way up to why specifically the designers of certain algorithms did it that way rather than another way.

- Advantages

  With this method being the most traditional way of learning a given subject, it has been tried and tested for many many years and works pretty much every time. With the additional bonus of having staff who will provide you any assistance you require over your few years of study, whether that would be on understanding a given topic or just to answer some of your questions. You also have ability to use any on-site facilities that the institute provide for you and this can be a massive help as it means you don't have to go out and spend money on specialist equipment to test something. You can just use the institutes.

- Disadvantages

  Even though this is the most traditional way to learn something in depth, like everything it does have flaws. For some people, learning this way just doesn't work well. You will spend most of your time learning about the theoretical side of the subject and will only look at the practical side on a few occasions. Couple this along with the fact that this route is very very expensive it makes it pointless for somebody that learns best by doing practical work.

1.3.2.2 2. Self Taught

Becoming a more and more popular way to learn about cryptography is to just teach yourself, its the route it took. This way allows you to quickly focus in on

exactly the aspect of cryptography that you find interesting and learn about it in depth.

- Advantages

  With the internet becoming more and more accessible to the average person it has made an abundance of learning resources available to anyone with a browser from the last decade. This given us an opportunity to learn more things than ever before and it is all completely free. There is plenty of information on cryptography that you can learn from, this includes YouTube series on every aspect of modern cryptography to research papers about an up and coming area of cryptography, like Quantum Cryptography. Combine this with the fact that you can learn at exactly the right pace for you and it is a very good option.

- Disadvantages

  The main drawback of this option is that you will need a device that allows you to connect to the internet, and you will need internet which does cost a fair amount. You also need the commitment to stick to it. Because you can go at your own pace, it does mean there is no one that will make you stick to learning about it. The entire responsibility of learning about the subject is in your own hands and for some this could be to much and they will just give up after about a week.

1.3.2.3 3. Learning on the Job

This option mixes the aspects of both the previously mentioned options. Some people will have already gotten a job in the technology industry before realizing how important cryptography is and will want to learn it so they know that they are doing there job right.

- Advantages

  With already working, you will be moved away from the theoretical and more the practical. This is great for some people. You will also be able to see exactly how these highly complicated concepts directly affect the way we use the internet. You may also have the opportunity through your employer to attend various courses and conferences that allow you to gain various certifications in cryptography and maybe even the broader InfoSec industry.

- Disadvantages

  The main disadvantage here is that you will need a job in a somewhat relevant area in the industry and in order to get the job in the first place you will probably have gone through options 1 and 2 anyway.

1.4 My Proposed Solution

1.4.1 End Goal

The end goal of my project is to provide a website that combines the best parts of Options 1 and 2 from the above list. It will teach the users everything from the core concepts of cryptography all the way to the very complicated systems that implement modern cryptography. This way people have the ability to learn as much as they would while in academia while getting the same freedom as if they were teaching themselves.

1.4.2 Proposed End Users

In the short term there my End Users will be a few students from the Computer Science class. Over the past few weeks while I have been learning Cryptography they have noticed and thought it was pretty cool and wanted to learn as well. But most of them just don't have the commitment to sit through hours of lectures like I did to learn the content so I though that I should build something that they can use.

Due to the content in this project, I will also be making it available to the public so anyone that really wants to learn about Cryptography will have the option to. This allows for a much larger range of people who will have the option to learn about Cryptography which everyone can see a being a good thing.

1.4.3 How the User will interact with the system

The main way that uses will interact with my system is quite simple. First they will need to create an account so they can record and keep track of there progress. They will then select a specific area to study. This is completely up the the user, they can work there way through chronologically going through each section or just jump to the middle and crack on with the hard stuff. Once in the selected section, they will read through the information content displayed on the screen. Once the user feels confident they understand that content, they can attempt to answer the sample questions at the bottom of the page. These questions will cover all content shown on that given page and will vary in difficulty. Once they have answered the questions they can check if they are right, this information will be shown in the report section, any questions they get wrong will be shown and the provided solutions as well so the user can see where they went wrong and learn from the mistakes made. Once they are happy with how they have performed in that given section they can move on to the next section repeating this process.

1.4.4 Technical Analysis of my System

1.4.4.1 Option 1: Classical Approach

My first idea about how I was going to attempt to build this system was arguably the 'classical' approach. By using the 3 core web technologies, (HTML CSS JS), I would have all the features that would be needed to fully achieve my goal.

- Advantages

  The biggest advantage of choosing this option would be that it has been around for decades. It is truly a tried and tested method. This means that not only does it have the flexibility that I would require, but if I were to run across any problems then I would just have to use the internet and chances are someone else has had that problem before and they would know how I could fix it and make it work. This is a massive perk over the other 2 choices as it means that even if I come across an abundance of problems, I will definitely be able to find a solution in a timely manner.

  An another advantage is that Javascript, that main language I would be using to 'code' the website, has many frameworks and extensions that I would be able to use freely. This allows me the ability to be a lot more ambitious about what I want to do as I wouldn't have to develop the majority of the core complex functions. I would just have to know how I need to implement them.

- Disadvantages

  The main disadvantage of this option for me is that personally I tend to stay away from this kind of work. I personally am much more of a back-end type developer. So I would definitely go for one of the other options.

  I also have much more experience with the back-end type languages and feel I would be spending the majority of my time learning the concepts and syntax of this option, whereas if I were to choose one of the other options I already know the syntax so could start working straight away.

1.4.4.2 Option 2: Modern Approach

This Option is the most 'modern' approach I could think of. This option considers all options with similarities to ASP.NET. I consider this the 'modern' approach as we are not using specific web technologies to create our website. We are just using a standard High Level language to do it, the main 2 contenders here are clearly C-Sharp and Java.

- Advantages

  The main advantage here is that by using a normal programming language, you have lots of flexibility. Because this is a normal programming

language you are not bound by the paradigms of normal web development. You can effectively create a Desktop application and with very little modifications, it will be possible to turn it into a Web application. Another advantage is that it is very easy to incorporate previously coded modules into your web application.

This also has the advantage that it suits my needs better as well. Since I have lots more experience with these types of languages, I would have much more time to spend on the actual web application rather than learning the language. This means I can get more done and have a better, more complete end result.

This option is also new technology, not so new that there is no documentation and help but not so old there is the same amount of help that Option 1 provides. Since it is in this mid-stage area, it will only be used more and more so looking at it in terms of what industry might need in a few years, this would give me a good opportunity to learn a technology that will make me more employable than the average person.

- Disadvantages

  The main disadvantage of this option is that since it revolves around fairly new technology, it will not be streamlined to the point of Option 1. What I mean by this is that it may take more effort to create the same thing from Option 1 using this method. This method may provide the best solution, but it may take a far longer time to get to the 'best' solution when compared to a 'client ready' solution from Option 1. The solution from Option 1 may not be as good as this one, but on a scale of 'first conception' to 'ready to ship', this Option is not the one to choose.

1.4.4.3 Option 3: The Hybrid

The final option is a sort of combination of the other Options but with a sort of twist. It will be using the C++ language, which is by no means the go to for creating web applications. It also includes using a 3rd party library which will provide the necessary core web technology functionality.

- Advantages

  The clear advantage of this option is that because it uses a 'mid' level language, that allows us to write very fast code. Rather than having to go through a framework and converting from source to bytecode to asm, we can go directly from the source to the asm. This is a very important factor as the the whole purpose of this project is to teach Cryptography and in a lot of situations, the key algorithms need to be fast. So, in that aspect there is no other option.

  This option is also my preferred option as C++ is a pure back-end language. I also have the most experience with this language therefore I would be most comfortable working with it. This also removes the need

to learn a completely new language to continue with the project. Thus eliminating a massive bottleneck in the process of development.

I would also consider this a very good option to make myself more employable in the future as lots of the older systems are written in C++ and in order to keep them running you need someone who is familiar with the language.

- Disadvantages

  The key disadvantage with this option is that it uses a not-so friendly language. Meaning that it takes much more effort to do the same task when compared to the other options. It also has the disadvantage that you need to learn a completely new library to do anything which can be very daunting to consider.

1.5 Requirements / Objectives / Limitations

This whole section is going to serve a check list of the requirements, objectives and limitations of my project. The requirements serve as a detailed check list for me to go over and use to see if my project has all the content I want it to have. The objectives serve as a more 'is my project complete' check list and that is what will be used when checking as to whether or not my project is complete per say. And finally, limitations are similar to objectives but are things that I would like to achieve but couldn't due a variety of reasons including but not limited to time constraints and complexity of the problem.

There are 2 main sections here relating to the User and System. Anything that relates the the User is about how the platform is designed and orientated around the user. When it comes to the System it is mainly about my implementation for the solution.

1.5.1 User Requirements

- Easy to read font
- Intuitive user interface
- Good flow through out the entirety of the website
- Content explained makes sense
- Hard / Complex areas explained simply
- Content should be easy to follow

1.5.2 System Requirements

1.5.2.1 Platform Requirements

A High Level look at what I want the platform to cover.

- Cryptography
  - Basic information about Cryptography
    - Private-Key Cryptography
    - Introduction to Public-Key Cryptography
      - Security Mechanisms
      - Authenticity of Public Keys
      - Important Public-Key Algorithms
      - Key Lengths and Security Levels
    - Explain the simple protocols
  - Overview of the types of Ciphers
    - Block Cipher
    - Stream Cipher
    - Encryption and Decryption using both types of ciphers
  - Overview of Hashing Functions
  - Brief History on Cryptography
    - Caesar Cipher
    - Enigma Machine
- Cryptanalysis
  - Very Basic overview of what Cryptanalysis involves
  - Brief History on Cryptanalysis

Going into more detail about what I want to teach specifically.

- Cryptography
  - Public-Key Cryptography
    - Cryptosystems based of the Discrete Logarithm Problem
      - Diffie-Hellman Key Exchange
      - The Discrete Logarithm Problem
      - The Elgamel Encryption Scheme
    - RSA Cryptosystem
    - Eliptic Curve
    - Digital Signatures
      - RSA Signature Scheme
      - Elgamel Digital Signature Scheme

            - Digital Signature Scheme (DSA)
            - Elliptic Curve Digital Signature Scheme (ELDSC)
        * Key Establishment
        * Message Authentication Codes (MACs)
            - MACs from Hash Functions
            - MACS from Block Ciphers CBC-MAC
            - Galois Counter Message Authentication Code (GMAC)
    - More detailed look into the common algorithms
        * Block Ciphers
            - Data Encryption Standard (DES)
            - Advanced Encryption Standard (AES)
        * Talk about the modes of operation for Block Ciphers
            - Electronic Codebook Mode (ECB)
            - Cipher Block Chaining (CBC)
            - Output Feedback Mode (OFB)
            - Cipher Feedback Mode (CFB)
            - Counter Mode (CTR)
            - Galois Counter Mode (GCM)
        * Stream Ciphers
            - Stream ciphers based off Linear Feedback Shift Register (LFSR)
        * Hashing Functions
            - MD5
            - SHA-1
            - SHA-2
            - SHA-3
    - Implementations of the ciphers shown above
    - Create theory questions that require the user to demonstrate their understanding of the various Cryptography sections

Additional Content that I would like to add if possible but not critical to the completion of the project

- Cryptography
    - Build a simulator for the enigma machine and show graphically how it works
    - Coding Problems
        * Design Custom coding problems that test the users cryptology knowledge
        * Create a custom development environment so the user can enter the solution to the problem on the website and it will be able to tell you if the answer is correct or not

- Cryptanalysis
  - Explain some of the common techniques used in cryptanalysis
  - Provide come example ciphers for the users to practice their cryptanalysis on

1.5.2.2 AES Implementation Requirements

- Key Expansion Implemented
- AddRoundKey Function Implemented
- SubstituteBytes Function Implemented
- ShiftRows Function Implemented
- MixColumns Function Implemented
- Key Schedule Implemented

1.5.2.3 DES Implementation Requirements

- Initial Permutation Implemented
- Feistal (F) Function Implemented
  - Key Expansion
  - Key Mixing
  - Substitution
  - Permutation
- Final Permutation Implemented
- Key Schedule Implemented

1.5.2.4 SHA-1 Implementation Requirements

1.5.3 System Objectives

1.5.3.1 Platform Objectives

I have split up this section into 3 different areas, Website Management, Website Content, User Interaction to make it easier to understand. It is also organized from most important to least important for each section. E.g. the higher up in the list the more important the item is.

1. Website Management
   (a) Website that I can launch and connect to from my Computer

(b) Buying a Domain for the Website

(c) Renting a VM for my Website to live in so it can be accessible to anyone

(d) Renting a SSL Certificate so I have HTTPS enabled for secure communication between the client and the server

2. Website Content

(a) Contains content on Symmetric Cryptography

(b) Contains theoretical questions on Symmetric Cryptography

(c) Contains content on Asymmetric Cryptography

(d) Contains theoretical questions on Asymmetric Cryptography

(e) Create online code-editor with standard code-editor features. E.g. Auto-complete, re factoring, a home made 'intellisense'

(f) Create questions that require the need to write code in order to solve them

(g) Review the Coded solutions and suggest improvements

(h) Create Visual simulation of various algorithms and cryptographic systems

i. AES and DES

ii. The Engima Machine

3. User Interaction

(a) Login system is implemented

(b) Authentication for users is implemented

(c) Relevant user information is stored in local database

(d) Email verification for accounts is implemented

(e) Store answers to questions into local database

(f) Analyse the users answers to questions and produce report on what they need to work on

(g)

1.5.3.2 AES Implementation Objectives

1. Key sizes supported

(a) 128-bit

(b) 192-bit

(c) 256-bit

2. Different Modes of Operation

(a) ECB (Electronic Code Book
(b) CBC (Cipher Block Chaining)
(c) OFB (Output Feedback Mode)
(d) CFB (Cipher Feedback Mode)
(e) CTR (Counter Mode)
(f) GCM (Galois Counter Mode)

1.5.3.3 DES Implementations Objectives

1. Different Modes of Operation
   (a) ECB (Electronic Code Book
   (b) CBC (Cipher Block Chaining)
   (c) OFB (Output Feedback Mode)
   (d) CFB (Cipher Feedback Mode)
   (e) CTR (Counter Mode)
   (f) GCM (Galois Counter Mode)

1.5.3.4 SHA-1 Implementation Objectives

1.5.4 Acceptable Limitations

1.5.4.1 Platform Limitations

1. Website Management
   (a) Limit to concurrent connections to the website
   (b) Making the website live to the public
   (c) Getting SSL/HTTPS setup for the website
2. Website Content
   (a) Online Code-editor does not support all the modern features
   (b) Visual simulations are not implemented
3. User Interaction
   (a) Analysis of user answers not fully finished

1.5.4.2 AES Implementation Limitations

1. Only 128-bit Key size is supported
2. Only configure ECB Mode of operation
3. Padding is not implemented

1.5.4.3 DES Implementation Limitations

1. Only configure ECB Mode of operation
2. Padding is not implemented

1.5.4.4 SHA-1 Implementation Limitations

1.6 Data Usage in the System

1.6.1 Data Security Overview

This will be covered more in depth in the Documented Design section but I feel it is important to briefly go over it here.

So for all important user critical information stored in the local database, the appropriate security measures will be implemented. Since the only critical user information is the passwords to the users account, before storing the password it will be hashed using a secure hashing algorithm, such as bcrypt, SHA2, SHA3, etc. The reason we hash it rather than encrypt it is that a hash is known as a 1-way function. You can never get the plaintext back from the ciphertext, in this case our plaintext is the password and the ciphertext is the password hash or digest.

1.6.2 Data Sources

$\frac{a}{b}$

1.6.3 Data Destinations

1.6.4 Data Volumes

1.6.5 Data Analysis

1.6.5.1 User Data

1.6.5.2 System Data

1.6.5.3 Temporary Data

1.6.6 Data Flow

Chapter 2

Documented Design

2.1 High-Level Overview of System

2.2 Project Versioning

2.2.1 File Structure

This section will talk about the general file structure I will be using and the style that I will be coding against. We will be using an example file, file˙structure.cc, which is shown below.

```
1  /**
2   * @file file_structure.cc
3   * @date 21/02/2018
4   * @version 0.01
5   *
6   * @brief A brief explanation of what is contained in this file and what ↩
        the purpose of the file is
7   *
8   * @author Jacob Powell
9   */
10
11
12 /**
13  * @class A
14  * @version 0.01
15  *
16  * @brief A brief explanation of what the purpose of the class is and ↩
        what it does
17  */
18 class A
19 {
20
21 };
22
23 /**
24  * @class TestClass
25  * @version 0.01
```

```
26  * @implements TestClass
27  *
28  * @brief A brief explanation of what the purpose of the class is and ↩
        what it does
29  */
30 class B : A
31 {
32 public:
33     /**
34      * @brief A brief explanation of what the function does
35      *
36      * More detailed explanation below
37      */
38     TestClass() = default;
39
40     /**
41      * @brief A brief explanation of what the function does
42      *
43      * @param f A description of the parameters is given here
44      * @param e
45      * @return Describes what the function returns
46      *
47      * More detailed explanation below
48      */
49     int C(int f, bool e);
50
51 private:
52     int f; /**< Detailed information on a member of the class */
53
54 };
```

2.2.1.1 File Header

Firstly we will be looking at the generic file header for the *.h* and *.cc* files used in my project. The file header is given below:

```
1 /**
2  * @file file_structure.cc
3  * @date 21/02/2018
4  * @version 0.01
5  *
6  * @brief A brief explanation of what is contained in this file and what ↩
       the purpose of the file is
7  *
8  * @author Jacob Powell
9  */
```

There are 5 key sections included in the file header,

- @file - this declared the name of the file, including its extension
- @date - this is the date of creation of the file

- @version - this documents what the version of the file is. It will change when there are small changes and when there are large changes. The significance of the change is shown by which number is changed, e.g. changing from version 0.01 to 0.1 is a more significant change than going from version 0.01 to 0.02
- @brief - this is where a brief description of what the files purpose is and what it contains
- @author - this is where the author of the files name is given

The reason I am including a header block is because when browsing through my project most people will not be able to meticulously go through all the code working out what it does exactly. The header block serves for the purpose of explaining what the code in the file does, giving the reader any key information like version, date of creation and author as soon as they look at the file. This way the reader doesn't have to read all the code, they can just read the header block and they will understand, on a high level, what that section of the code does.

2.2.1.2 File Commenting

Throughout the project I will be sticking to a particular commenting style, which applies rules to how document my code. This section explains how I comment and what style I use for each scenario.

The first area we will look at is how we document classes, their members and their methods. We will look at the example class B and in our file˙structure.cc example file. The class extract is shown below.

Classes An extract of our example classes is given below:

```
1  /**
2   * @class A
3   * @version 0.01
4   *
5   * @brief A brief explanation of what the purpose of the class is and ↩
        what it does
6   */
7  class A
8  {
9
10 };
11
12 /**
13  * @class TestClass
14  * @version 0.01
15  * @implements TestClass
16  *
17  * @brief A brief explanation of what the purpose of the class is and ↩
        what it does
```

```
18  */
19 class B : A
20 {
21
22 };
```

A class will contain 4 sections:

- @class - The name of the class, the name should also always use Camel-Casing
- @version - The current version of the class
- @implements - If the class inherits from another class then the name of the inherited class is given here
- @brief - This provides a short explanation of what the class does and what its purpose is

Methods An extract of class method documentation is given below:

```
1  class B : A
2  {
3  public:
4      /**
5       * @brief A brief explanation of what the function does
6       *
7       * More detailed explanation below
8       */
9      TestClass() = default;
10
11     /**
12      * @brief A brief explanation of what the function does
13      *
14      * @param f A description of the parameters is given here
15      * @param e
16      * @return Describes what the function returns
17      *
18      * More detailed explanation below
19      */
20     int C(int f, bool e);
```

- @brief - Explains what the method does
- @param - If the method takes parameters then these will be declared and explained here
- @return - If the method returns something this explains what that is
- Then after these tags a more detailed explanation of what happens in the method will be given below

Members An extract of class member documentation is given below

```
class B : A
{
private:
    int f; /**< Detailed information on a member of the class */

};
```

- Next to the member use the style `/**< */` to describe what its purpose in the class is

2.2.2 Version Changelog

Lets look at the file header again, it is given below:

```
/**
 * @file file_structure.cc
 * @date 21/02/2018
 * @version 0.01
 *
 * @brief A brief explanation of what is contained in this file and what ↩
    the purpose of the file is
 *
 * @author Jacob Powell
 */
```

As you can see there is a *@version* tag. For every data structure and file in the project there will the a *@version* tag which will contain the most up to date version of the file / data structure. This allows me to easily keep track of how each element of my project is progressing and evolving. The behavior of this tag is explained in *Section 2.2.1.1*.

2.2.3 Off-Site Backups

When developing a project of this scale it is crucial that you don't just *keep all your eggs in one basket* per say. What I mean by this is that you don't want a single point of failure, if I finished my project and then a week before the submission date my laptop died and I couldn't recover the data from my hard-drive then I would not be able to submit my project and I would fail.

For this reason I will be using Github to store daily backups of my projects code and documentation. This way, even if my laptop dies and I have no way of recovering the data I would have only lost a days worth of work, rather than a few months of work.

This will not actually affect the end result of my project but I do feel it is important to talk about it as it is probably the most important step of all since if something goes wrong, you can recover, if you don't you cant.

2.3 Bespoke Algorithm Design

2.4 Cryptographic Algorithm Design

2.4.1 Galois Fields for AES

In AES, every single operation that is used is based on Finite Fields. This section will give you a brief introduction to what Finite Fields are and how they are used.

2.4.1.1 Introduction to Finite Fields

It should be stated now that the term *Finite Field* means the same thing as the term *Galois Field*. In Abstract Algebra there are 3 basic structures, the group, the ring, and the finite field.

Groups Definition A group is a set of elements G together with an operation $\cdot$ which contains 2 elements of G. A group has the following properties:

1. The group operator $\cdot$ is closed. That is for all $a, b \in G$, it holds that $a \cdot b = c \in G$
2. The group operation is associative. That is, $a \cdot (b \cdot c) \equiv (a \cdot b) \cdot c$ for all $a, b, c \in G$
3. There is an identity element $1 \in G$ such that $a \cdot 1 = 1 \cdot a = a$ for $a \in G$
4. There is an inverse element for all elements in G. That is for $a \in G$, there must be a element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$
5. A group is Abelian (or commutative) if, $a, b \in G$, $a \cdot b \equiv b \cdot a$

Roughly speaking a group is set with one operation and its corresponding inverse operation. If this operation is addition then the inverse operation will be subtraction, if the operation is multiplication then the inverse will be division. If we need a structure that can hold all 4 operations, that is where we will use Finite Fields (Galois Fields). It should be noted that it is common to denote a Finite Field as FF and a Galois Field as GF.

Finite Field Definition A field F is a set of elements with the following properties:

1. All elements of F for an additive group with the group operation "$+$" and the identity element 0.
2. All elements of F, except the identity element 0, form a multiplicative group with the group operation "$\times$" and the identity element 1.

3. When the two group operations are mixed, the distributivity law holds, e.g. for all $a, b, c \in F : a(b + c) = (ab) + (ac)$

In cryptography we are almost always interested in fields with a finite number of elements which we intuitively name Finite Fields or Galois fields. The number of elements in the field is called the *order* or *cardinality* of the field. It should be noted that a field with order m only exists if m is a prime power, e.g. $m = p^n$ where n is a positive integer and p is a prime number. p is called the characteristic of the finite field. This implies that there are finite fields with 11 elements or 81 elements, since $81 = 3^4$ or with 256 elements (since $256 = 2^8$ and 2 is a prime). There is no finite fields with 12 elements since $12 = 2^2 \cdot 3$ and 12, meaning 12 is not a prime power.

Prime Fields The most intuitive examples of FF are fields of prime order, fields with $n = 1$. Elements of the $GF(p)$ can be represented by the integers $0, 1, ..., p - 1$. The two operations are modular integer addition and multiplication modulo p. This means that if we consider the integer ring $\mathbb{Z}_m$ and m happens to be prime then $\mathbb{Z}_m$ is not only a ring but also a finite field.

In order to do arithmetic in a prime field we have to follow the rules for integer rings: Addition and multiplication are done modulo p, the additive inverse of any element a is given by $a + (-a) = 0 \bmod p$, and the multiplicative inverse of any non-zero element a is defined as $a \cdot a^{-1} = 1$

We can look at the example of the prime field $GF(2) = \{0, 1\}$, which is the smallest finite field that can exist. The additive and multiplicative tables for the GF are listed below.

+	0	1
0	0	1
1	1	0

Table 2.1: Additive Table for $GF(2)$

×	0	1
0	0	0
1	0	1

Table 2.2: Multiplicative Table for $GF(2)$

This is quite cool as it shows that addition in $GF(2)$, modulo 2 addition, is equivalent to and XOR Gate. It also shows us that multiplication in $GF(2)$ is equivalent to the logical AND Gate. This field, $GF(2)$ is very important for AES.

2.4.1.2 Extention Fields $GF(2^m)$

AES uses a FF of 256 elements and is denoted as $GF(2^8)$. This field was chosen as each of the elements in this field can be represented as exactly one byte. It becomes increasing important as the S-Box and MixColumn transformations treat every byte of the internal data path as an element of the field $GF(2^8)$ and manipulates the data by performing arithmetic in this finite field.

As we said before if the order of a FF is not prime, which is clearly the case here (2^8 is clearly not a prime number) the addition and multiplication operators can not be represented by addition and multiplication of integers modulo 2^8. Fields with $m > 1$ are called *extension fields*. In order for us to have the ability of working with these fields we need the following:

1. A different notation for the field elements
2. Different rules for when we perform arithmetic operations with the elements

So for the elements of these finite fields, we represent them as *polynomials* with coefficients and that when we compute with these we perform a certain type of *polynomial arithmetic*. The polynomials have a maximum degree of $m - 1$, so that there are m coefficients in total for every element. In the field $GF(2^8)$, which is the AES FF, each element $A \in GF(2^8)$ is represented as:

$$A(x) = a_7x^7 + a_6x^6 + ... + a_1x + a_0 \text{ where}$$
$$a_i \in GF(2) = \{0, 1\}$$

It is very important to realise that every element of $GF(2^8)$ can also be stored as an $8 - bit$ vector:

$$A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$$

This means we do not have to store the factors x^7, x^5, etc as it is clear from the bit positions to which power x^i each coefficients belongs.

2.4.1.3 Addition and Subtraction in $GF(2^m)$

This is actually really easy as we just follow the basic polynomial rules of addition and subtraction, we just add or subtract coefficients with equal powers of x. This is mathematically shown below, Let $A(x), B(x) \in GF(2^m)$. The sum of the two elements is then computed as the following:

$$C(x) = A(x) + B(x) = \sum_{i=1}^{m-1} c_i x^i$$

where $c_i \equiv a_i + b_i \bmod 2$

and the difference between the 2 pairs is computed as the following:

$$C(x) = A(x) - B(x) = \sum_{i=1}^{m-1} c_i x^i$$

$$\text{where } c_i \equiv a_i - b_i \equiv a_i + b_i \bmod 2$$

Since we perform modulo 2 addition with the coefficients, addition and subtraction are the same thing. An example of this is given below:

$$\begin{aligned} A(x) &= x^7 + x^6 + x^4 + 1 \\ B(x) &= x^4 + x^2 + 1 \\ C(x) &= A(x) + B(x) = x^7 + x^6 + x^2 \end{aligned}$$

Note if we were to work out the difference between the two polynomials $A(x)$ and $B(x)$ it would be the same as $C(x)$.

2.4.1.4 Multiplication in $GF(2^m)$

Multiplication in $GF(2^8)$ is the core operation in the MixColumn transformation in AES. Firstly, 2 elements of $GF(2^8)$ are multiplied using standard polynomial rules:

$$\begin{aligned} A(x) \cdot B(x) &= (a_{m-1}x^{m-1} + ... + a_0) \cdot (b_{m-1}x^{-1} + ... + b_0) \\ C'(x) &= c'_{2m-2}x^{2m-1} + ... + c'_0 \end{aligned}$$

Where:

$$\begin{aligned} c'_0 &= a_0 b_0 \bmod 2 \\ c'_1 &= a_0 b_1 + a_1 b_0 \bmod 2 \\ &\vdots \\ c'_{2m-2} &= a_{m-1} b_{m-1} \bmod 2 \end{aligned}$$

Realize that all coefficients a_i, b_i and c_i are elements of $GF(2)$ and that coefficients arithmetic is performed in $GF(2)$. In general the product polynomial, $C(x)$, will have a degree higher than $m - 1$ and will thus have to be reduced. The idea is similar to what we would do in prime fields: in $GF(p)$, we multiply the two integers, divide the result by a prime and consider only

the remainder. In extension fields the product polynomial $C(x)$ is divided by a special polynomial and we consider only the remainder after the polynomial division. The special polynomials are called irreducible polynomials and we need them for the module reduction. These polynomials are roughly comparable to prime numbers, basically their only factors are 1 and the polynomial itself. A mathematical description of Extension Field multiplication is given below:

Let $A(x), B(x) \in GF(2^m)$ and let

$$P(x) \equiv \sum_{i=0}^{m} p_i x^i, p_i \in GF(2)$$

be an irreducible polynomial. Multiplication of the two elements $A(x), B(x)$ is given as

$$C(x) \equiv A(x) \cdot B(x) \bmod P(x)$$

This means that we need a irreducible polynomial $P(x)$, of degree m and with coefficients from $GF(2)$, for every field $GF(2^m)$. Not all polynomials are reducible much like not every number is a prime. We only need to know the irreducible polynomial for AES; it is given below

$$P(x) = x^8 + x^4 + x^3 + x + 1$$

This polynomial is a part of the specification for AES.

Putting this all together, lets say that we have the 2 polynomials $A(x)$ and $B(x)$ where $A(x) = x^3 + x^2 + 1$ and $B(x) = x^2 + x$ and we want to multiply them in the $GF(2^4)$. The irreducible polynomial for this GF is given as $P(x) = x^4 + x + 1$.

First we have to work out the plain polynomial multiplication,

$$C'(x) = A(x) \cdot B(x) = x^5 + x^3 + x^2 + x$$

We can now reduce $C'(x)$ using the standard polynomial division method, but it can sometimes be easier to reduce each of the leading terms x^4 and x^5 individually.

$$\begin{aligned} x^4 &= 1 \cdot P(x) + (x + 1) \\ x^4 &\equiv x + 1 \bmod P(x) \\ x^5 &\equiv x^2 + x \bmod P(x) \end{aligned}$$

Now we just substitute that back into our $C'(x)$ expression and we will get our result for the multiplication of $A(x)$ and $B(x)$

$$C(x) \equiv x^5 + x^3 + x^2 + x \bmod P(x)$$
$$C(x) \equiv (x^2 + x) + (x^3 + x^2 + x) = X63$$
$$A(x) \cdot B(x) \equiv x^3$$

We need to make sure we do not confuse multiplication in $GF(2^m)$ with integer multiplication, especially if we are concerned with software implementations of Galois fields.

2.4.1.5 Inversion in $GF(2^m)$

Inversion in $GF(2^8)$ is the core operation in the Byte Substitution Layer of AES which contains the S-Boxes. For a given FF $GF(2^m)$ and the corresponding irreducible polynomial $P(x)$, the inverse A^{-1} of a nonzero element $A \in GF(2^m)$ is defined as:

$$A^{-1} \cdot A(x) = 1 \bmod P(x)$$

For small fields, in practice fields with 2^16 or fewer elements, look-up tables which contain the precomputed inverses of all finite elements are often used. Figure 2.1 shows the multiplicative inverse for $GF(2^8)$ used in AES.

	Y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
X 8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

Figure 2.1: The multiplicative inverse table of $GF(2^8)$ for bytes xy used within the AES S-Box

As an alternative to using lookup tables, it is possible to compute the inverses of all the elements. The main algorithm uses for this task is the Extended Euclidean Algorithm (EEA).

2.4.2 AES Internals

2.4.2.1 Structure of AES

It should be noted that the 128-bit data path that runs through the algorithm is split up into 16-bytes, this is why many people would consider AES a byte oriented block cipher as it works in pure bytes. So our structure diagram from Figure 1.3 can be adapted to the structure shown in Figure 2.2. We also noted before that the algorithm operates on a 4×4 matrix of bytes, Figure 1.2, and this remains true so keep that in mind when performing any operations, this becomes especially important in the ShiftRows and MixColumns steps.

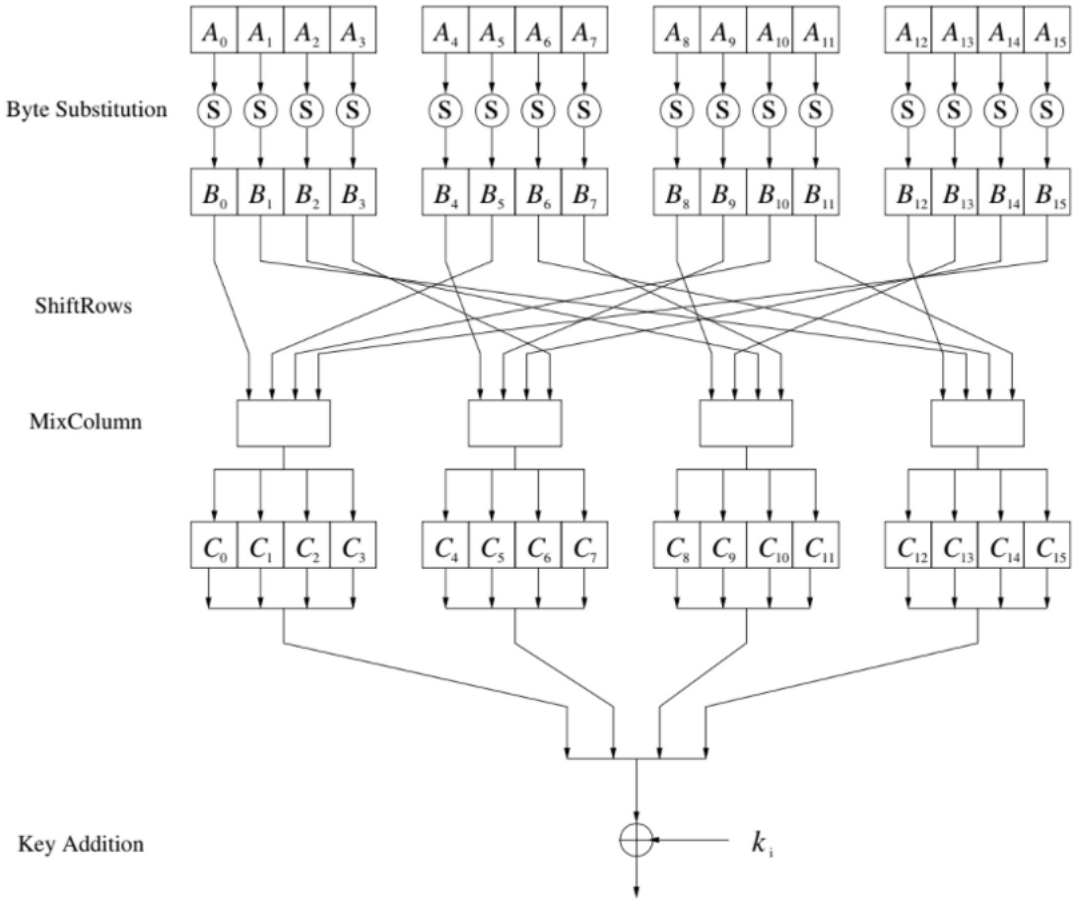

Figure 2.2: AES Byte Level Overview

The rest of this section will be looking into how the different components of each round work, these being the ByteSub Layer, ShiftRows Layer, MixColumns Layer and Key Addition Layer.

2.4.2.2 Byte Substitution Layer

The first layer in AES is the Byte Substitution Layer. There are 16 parallel S-Box which all take 8-bits as input and 8-bits as output. All 16 S-Box are

identical with is different to DES as that cipher uses 8 unique S-Boxes. Each state Byte A_i is replaced, substituted, with another Byte, B_i. The process is shown below:

$$S(A_i) = B_i$$

The S-Box is the only non-linear section of AES. This basically means that

$S(A) + S(B) \neq S(A + B)$ With 2 Byte States A and B.

The S-Box substitution is a 1-to-1 mapping, basically all of the $2^8 = 256$ elements are mapped to one output element. This is good as it allows us to uniquely reverse the S-Box, which is key to the decryption process. In software, the S-Box is usually setup as a 256 Byte lookup table with fixed entries. The AES S-Box is given below for an input byte (YX), with (YX) representing its hexadecimal value in each of its respective columns.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.3: The AES S-Box

So lets look at an example, say we have the Byte State $A_i = (3b)_{hex}$, then we apply the S-Box and get the following result:

$$S(A_i) \equiv S((3b)_{hex}) \equiv (e2)_{hex}$$

If we were to look at this on a bit level, the point of interest for the cryptography of the cipher, then the substitution becomes:

$$S(A_i) \equiv S(00111011) \equiv (11100010)$$

Even though the S-Box has a 1-to-1 mapping, it does not have any fixed points. This means that there are no input values A_i that make $S(A_i) = A_i$. Even the zero-input state is not a fixed point, i.e $S((00)_{hex}) = S(00000000) = (01100011)$.

Mathematical Description of S-Box

ADD THIS IN LATER

2.4.2.3 ShiftRows Layer

The ShiftRows Layer does exactly what its name implies. It cyclically shifts the rows of the byte matrix, see Figure 1.2. It shifts the first row by 0 bytes, the second row by 1 byte to the left, the third row 2 bytes to the left and the fourth row by 3 bytes to the left. If we take out original byte matrix, also shown in Figure 1.2, shown below:

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

After the ShiftRows Layer, the matrix would look like this, shown below.

b_0	b_4	b_8	b_{12}
b_5	b_9	b_{13}	b_1
b_{10}	b_{14}	b_2	b_6
b_{15}	b_3	b_7	b_{11}

2.4.2.4 MixColumns Layer

The MixColumns Layer is a linear transformation step which mixes the column of the state matrix. The combination of the ShiftRow and MixColumn layers makes it possible that after only three rounds, every byte of the state matrix depends on all 16 plaintext bytes. We denote the MixColumn layer with input matrix B and output matrix C as the following:

MixColumn(B) = C

Where B is the state matrix after the ShiftRows layer. Now we take each column as a vector and is then mulitplied by a fixed 4×4 matrix. This matrix is constant. Multiplication and addition of the coefficients is done in $GF(2^8)$. The computation of the first four output bytes is shown below.

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

The second column of C, (C_4, C_5, C_6, C_7) is computed by multiplying the same constant matrix by (B_4, B_9, B_{14}, B_3), the third and fourth columns of C ad heed to this as well.

We know that each state byte, C_i and B_i are 8-bit values that represent a value in $GF(2^8)$. All arithmetic done is performed in this layer is formed in $GF(2^8)$. The constants in the matrix are hexadecimal numbers, representing the corresponding values in the Galois Field. So 01_{hex} is 00000001_{bin} which corresponds to the $GF(2^8)$ element 1, 02_{hex} is 00000010_{bin} corresponds to the $GF(2^8)$ polynomial x and 03_{hex} is 00000011_{bin} corresponds to the $GF(2^8)$ polynomial $x+1$.

The additions here are performed in $GF(2^8)$ so are simple XOR operations with the 2 different bytes. For the multiplications, the constants $01, 02, 03$ were chosen as they are very easy to set up in software. Multiplication with 01 is just multiplication by the identity which doesn't require a given operation. Multiplication by 02 and 03 are more complicated, therefore 2 256 byte lookup tables are used. You could do multiplication by 02 as a multiplication by x, which is just a left shift of 1, and then a modular reduction by the irreducible polynomial of $GF(2^8)$, $P(x) = x^8+x^4+x^3+x+1$. Multiplication by 03 is similar, it is just a left shift of 1 and then adding the original value followed by modular reduction of $P(x)$ again.

As an example, lets take the input state array, $B = (25, 25, ..., 25)$. Since we are only doing the first column this only involves 2 multiplications, by 02 and 03.

$$\begin{aligned} 02 \cdot 25 &= x \cdot (x^5 + x^2 + 1) \\ &= x^6 + x^3 + x \\ 03 \cdot 25 &= (x + 1) \cdot (x^5 + x^2 + 1) \\ &= (x^6 + x^3 + x) + (x^5 + x^2 + 1 \\ &= x^6 + x^5 + x^3 + x^2 + x + 1 \end{aligned}$$

Since the order of our intermediate values, $x^6 + x^3 + x$ and $x^6 + x^5 + x^3 + x^2 + x + 1$, do not have an order of above 8 we don't need to perform modular reduction with $P(x)$, the next step is to add together all the components of the multiplication.

$$\begin{aligned} 02 \cdot 25 &= 1x^6 + 0x^5 + 0x^4 + 1x^3 + 0x^2 + 1x + 0 \\ 03 \cdot 25 &= 1x^6 + 1x^5 + 0x^4 + 1x^3 + 1x^2 + 1x + 1 \\ 01 \cdot 25 &= 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 0x + 1 \\ 01 \cdot 25 &= 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 0x + 1 \end{aligned}$$

$$C_i = 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 0x + 1$$

This is where $i = 0, 1, 2, ..., 15$. This means our output state is the value of the $GF(2^8)$ polynomial $x^5 + x^2 + 1$ which is 25. So $C = (25, 25, ..., 25)$.

2.4.2.5 Key Addition Layer

The key addition layer takes 2 inputs, the current 16-byte state matrix and a sub key which is also 16-bytes. Then the 2 inputs are combined using the XOR operation. The particular sub keys that are derived are explained in the next section, *Section 2.4.2.6*.

2.4.2.6 AES Key Schedule

The Key Schedule takes our original 128/192/256 bit key and will then derive the subkeys needed for the algorithm. At the beginning and end of AES a XOR addition is applied to the subkey, this process is sometimes referred to as key-whitening. The number of subkeys required is the number of rounds plus one as we need an extra subkey for the key-whitening. A table with the key lengths, round numbers, and subkeys is shown below.

$N_k(bits)$	N_r	N_{sk}
128	10	11
192	12	13
256	14	15

Table 2.3: Repsective Key Lengths, Rounds Numbers, Subkey Numbers

The AES Key Schedule is word oriented, where 1 word = 32-bits. Sub keys are stored in an expansion array called W that consists of words. There are 3 different key schedules for the 3 different key sizes.

128-bit Key Schedule The 11 subkeys are computed and stored in an array with elements $W[0], W[1], ..., W[43]$.. Figure 2.4 shows exactly how the subkeys for AES with a 128-bit key are computed.

The first subkey, k_0, is just the original key used when initalizing the algorithm. So elements $W[0], .., W[3]$ contain our original $128 - bit$ key. Then, the left most word of a subkey $W[4i]$, where $i = 1, 2, ..., 10$, is computed using the following formula:

$$W[4i] = W[4(i-1)] \oplus g(W[4i-1])$$

With the function g here being a non-linear function with a $4 - byte$ input and a $4 - byte$ output. The remaining elements of W are calculated using the following formula:

$$W[4i+j] = W[4i+j-1] \oplus W[4(i-1)+j]$$
Where $i = 1, 2, 3, ..., 10$ and $j = 1, 2, 3$.

The function $g()$ rotated our $4 - byte$ input, it applies a S-Box substitution and adds a round coefficient, RC, to it. $RC's$ value is an element from $GF(2^8)$,

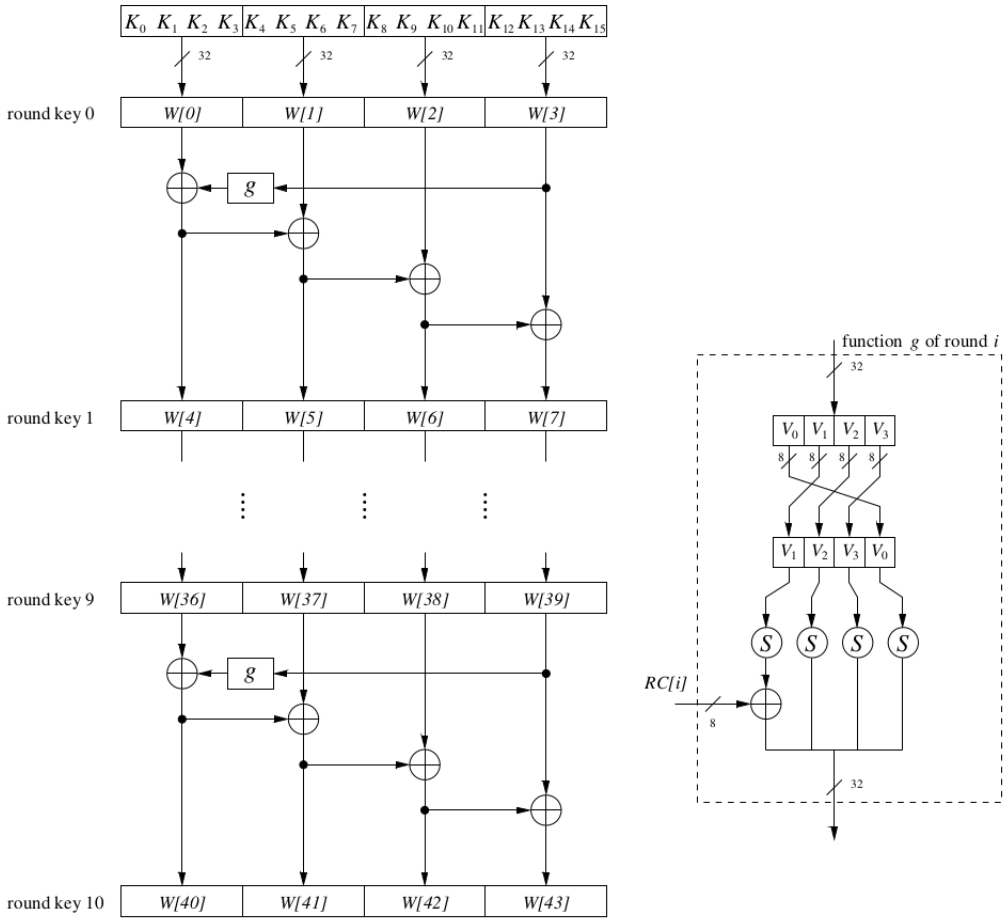

Figure 2.4: 128-bit Key Schedule for AES

i.e. an 8-bit value. This value is only added to the left most byte in the function $g()$. The RC vary from round to round according to the following rule:

$$
\begin{aligned}
RC[1] &= x^0 = (00000001)_2, \\
RC[2] &= x^1 = (00000010)_2, \\
RC[3] &= x^2 = (00000100)_2, \\
&\vdots \\
RC[10] &= x^9 = (00110110)_2,
\end{aligned}
$$

This function $g()$ has 2 purposes, its first is to add non-linearity to the key schedule and the second is to remove symmetry from the AES cipher. Both of these are required as it protects from some attacks against block ciphers.

192-bit Key Schedule With a 192-bit key, we need 13 subkeys, which means our array W will have 52 words. The way we compute all of the 13 words, and thus the 52 words, is very similar to the way we compute the subkeys for a 128-bit key, this is shown is Figure 2.5. There are 8 iterations of the Key Schedule, where each iteration computes 6 new words for the subkey array W. We need 8 round coefficients, $RC[i]$, for use in our $g()$ function. These round constants are computed the same way as in a 128-bit key case.

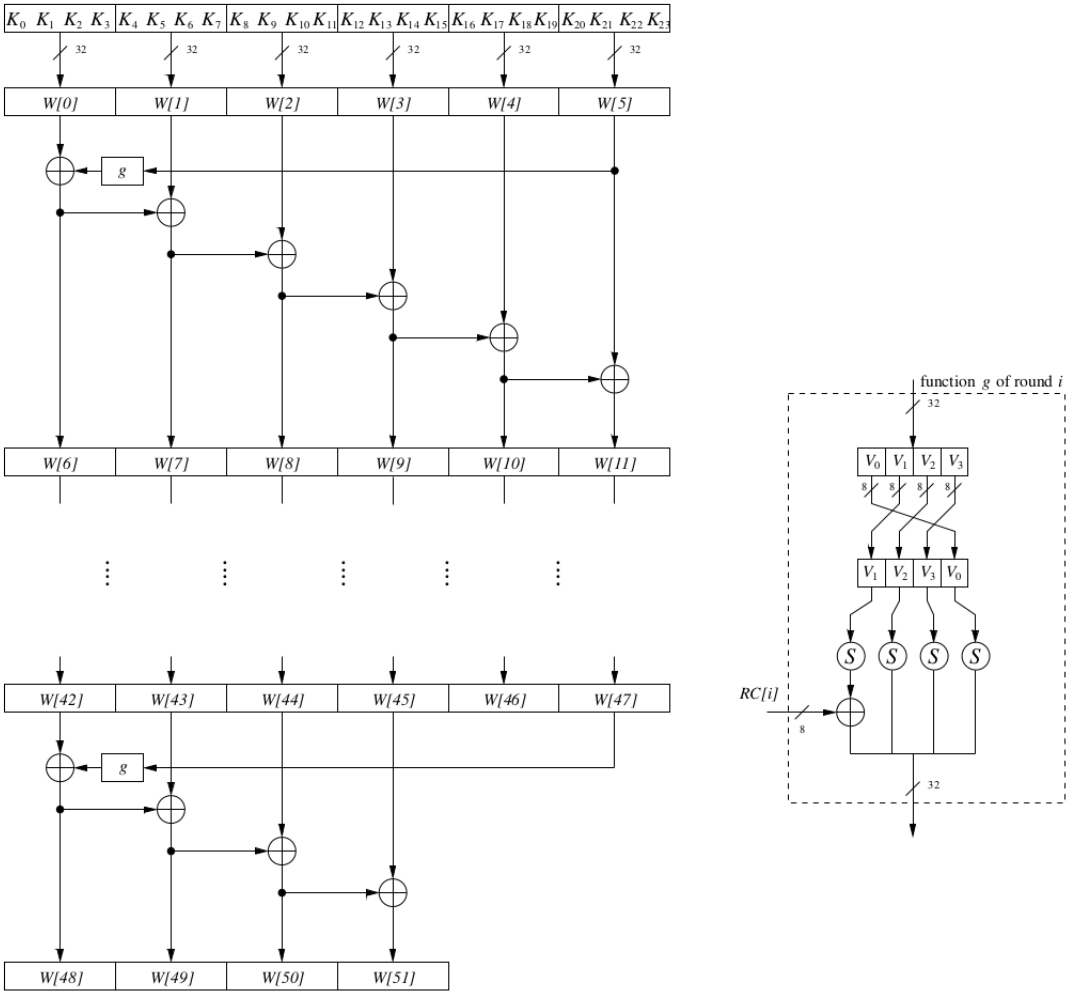

Figure 2.5: 192-bit Key Schedule for AES

256-bit Key Schedule With a 256-bit key. we need 15 subkeys, meaning our array W will contain 60 words. The way we compute all of the 13 words, and thus the 52 words, is very similar to the way we compute the subkeys for a 128-bit key, this is shown is Figure 2.6. There are 7 iterations of the Key Schedule, where each iteration computes 8 new words for the subkey array W. We need 7 round coefficients, $RC[i]$, for use in our $g()$ function. These round constants are computed the same way as in a 128-bit key case. The

key schedule also introduces a new function $h()$, which takes a 4-byte input and output. $h()$ applies the S-Box to each input byte.

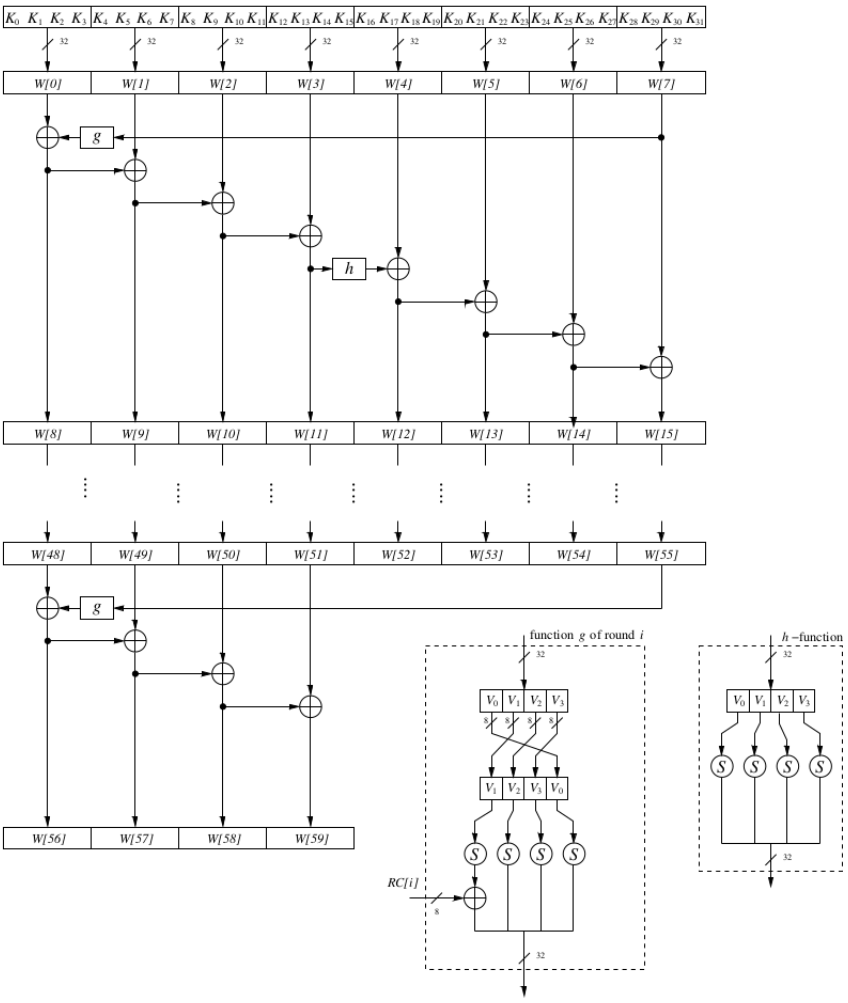

Figure 2.6: 256-bit Key Schedule for AES

2.4.3 Decryption in AES

Unlike DES, because AES is not based on a feistal network each layer needs to be inverted in order to decrypt a message that was encrypted using AES. This means that the S-Box, ShiftRows, MixColumns need to be inverted. For the AddRoundKey, this words slightly differently as for this all we need to do is reverse the order of the key schedule. So for the first round we use the last subkey, the second round we use the second-last subkey. The diagram below, Figure 2.7, illustrated this process.

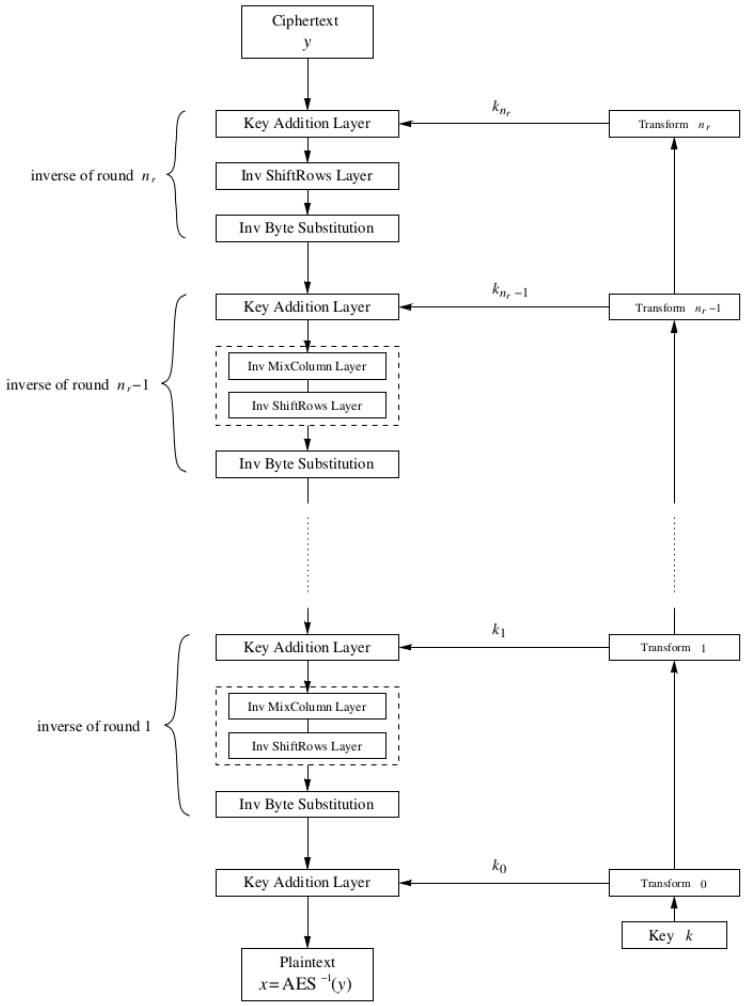

Figure 2.7: The Decryption process for AES

2.4.3.1 Inverse MixColumns Layer

After the addition of the Sub key, the inverse MixColumns step is applied to the state matrix, the exception is on the first decryption round where MixColumns is not applied. We inverse the MixColumn operation by just using the inverse matrix of the MixColumn operation. We have our 4×4 input state (C_0, C_1, C_2, C_3) and we just multiply this by the Inverse Matrix to get our output state (B_0, B_1, B_2, B_3).

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

Then in order to calculate (B_4, B_5, B_6, B_7) we just use the input (C_4, C_5, C_6, C_7). B_i and C_i are elements from $GF(2^8)$ and also the constants in our matrix are

elements of $GF(2^8)$. It should also be said that additions in the vector-matrix multiplication are bitwise XOR's.

2.4.3.2 Inverse ShiftRows Layer

In order to reverse the Shift Rows Layer all we have to do is shift the elements in the state matrix the opposite direction. The first row is not affected as it is also not affected in the normal ShiftRows layer. With our state matrix $B = (B_0, B_1, B_2, B_3, ..., B_15)$:

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

The Inverse Shift Rows operation would give us the following state matrix:

b_0	b_4	b_8	b_{12}
b_{13}	b_1	b_5	b_9
b_{10}	b_{14}	b_2	b_6
b_7	b_{11}	b_{15}	b_3

2.4.3.3 Inverse ByteSubstitution Layer

The inverse S-Box is applied when decrypting with AES. Since the AES S-Box is bijective, one-to-one mapping, it is possible to create an inverse S-Box such that:

$$A_i = S^{-1}(B_i) = S^{-1}(S(A_i))$$

The values given in the Inverse S-Box are given in Figure 2.8.

2.4.3.4 Decryption Key Schedule

Since the first round of the decryption round needs the last subkey, the second round of decryption needs the second-last subkey, etc, etc. So we need to supply the subkeys in reverse order for the decryption rounds. This can generally be done by computing the 11 or 13 or 15 subkeys and storing them. This adds some latency to the decryption of the algorithm when compared to the encryption rounds.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.8: The AES Inverse S-Box

2.4.4 DES

2.4.4.1 Structure of DES

2.4.4.2 Internals of DES

2.4.4.3 Operations with DES

2.5 Bespoke Data Structures

2.6 Bespoke Database Design

2.6.1 User Details

a

2.6.2 User Answers

a

2.6.3 Authentication Information

a

2.6.4 Authentication Identity

a

2.6.5 Authentication Token

a

2.6.6 Question Details

a

2.7 User Interface Design

a

2.7.1 Header

a

2.7.2 Information Window

a

2.7.3 Login Page

a

2.7.4 Register Page

a

2.8 System Security

2.8.1 Security Requirements

2.8.2 User Details Storage

2.9 External Library's

2.9.1 Wt

Wt, said *Witty*, is a Web framework for the C++ programming language. It provides the basic widgets and building blocks needed to create A Web Appli-

cation but also has a lot of other really useful features. This include but are not limited to:

- Client Side Optimized

  By using a signal-slot system, we don't have to worry about dealing with the AJAX requests. We simply connect a widget to the callback function of the server and its done. It also will use whatever technology is available for use in the communication, normally AJAX or WebSockets, but if will fall back into full html page reloads if javascript is not available. This allows a Wt Application to be accessible to any browser

- Built-In Security

  By default, Wt only allows the visible and enable widgets to be interacted with. It also helps prevent CSRF attacks by not storing any of the session information in cookies. Because of the Widget abstraction it discourages the use of raw html in the application which helps to prevent again XSS attacks. SQL Injection attacks are stopped by encouraging the user to use prepared SQL statements when accessing the database. Wt also includes an authentication and registration system with support for OAuth providers like Google and Facebook

- Object Relational Mapping Library

  This maps C++ classes to tables in my database using Wt::Dbo, a ORM that only requires pure C++. It does not depend on preprocessor magic or code generation.

2.9.2 Crypto++

Crypto++ is a C++ library that contains implementations for a huge amount of Cryptographic algorithms.

Chapter 3

Technical Solution

3.1 AES Implementation

3.1.1 aes_implementation.h

```
1  /**
2   * @file aes_implementation.h
3   * @date 14/01/2018
4   *
5   * @brief This contains the class definition for my AES Implementation
6   *
7   * @version 0.10
8   * @author Jacob Powell
9   */
10
11 #include <cstdint>
12 #include <string>
13
14 typedef uint8_t byte; /**< This is a simple typedef that makes my life ↩
       easier */
15
16 /**
17  * @enum AESKeyLengths
18  *
19  * @version 0.01
20  * @brief This enum contains the possible key lengths that the AES ↩
        algorithm can use.
21  */
22 enum AESKeyLengths{
23     AES128,
24     AES192,
25     AES256
26 };
27
28
29 /**
30  * @class AESImplementation
31  *
```

```
32  * @version 0.01
33  * @brief This class contains my implementation for the AES encryption ↩
        algorithm
34  */
35 class AESImplementation {
36 public:
37     /**
38      * @brief Setting constructor with no arguments to delete
39      */
40     AESImplementation() = delete;
41 
42     /**
43      * @brief Default constructor which sets the encryption 128 bit key ↩
            to be used
44      *
45      * @param key The encryption key passed into the algorithm
46      */
47     explicit AESImplementation(AESKeyLengths keyLength);
48 
49     /**
50      * @brief This method provides the encryption functionality for the ↩
            AES algorithm
51      *
52      * @param input The plaintext that wants to be encrypted
53      * @param output The ciphertext that has been encrypted
54      * @param key The algorithm key
55      */
56     void encrypt(const byte input[], byte output[], const byte key[]);
57 
58     /**
59      * @brief This method provides the decryption functionality for the ↩
            AES Algorithm
60      *
61      * @param input The encrypted ciphertext
62      * @param output The decrypted plaintext
63      * @param key The algorithm key
64      */
65     void decrypt(const byte input[], byte output[], const byte key[]);
66 
67 private:
68 
69     /**
70      * @brief This provides the implementation for the KeyExpansion part ↩
            of the algorithm
71      *
72      * @param key The original 128 bit key
73      * @param expandedKey The expanded key used for the algorithms round
74      */
75     void KeyExpansion(const byte key[], byte expandedKey[]) const;
76     static void KeyExpansionCore(byte roundNumber, const byte keyIn[4], ↩
           byte keyOut[4]);
77 
78     void AddRoundKey(byte state[4][4], byte roundKey[4][4]);
79 
80     void SubBytes(byte state[4][4]);
```

```
81      void ShiftRows(byte state[4][4]);
82      void MixColumns(byte state[4][4]);
83
84      void InverseSubBytes(byte state[4][4]);
85      void InverseShiftRows(byte state[4][4]);
86      void InverseMixColumns(byte state[4][4]);
87
88      void outputState(std::string prefix);
89      void outputRoundKey();
90
91      byte _state[4][4]; /**< This will hold the current state of the ↩
            algorithm */
92      byte _round_key[4][4]; /**< This will hold the round key */
93
94
95      int _number_of_rounds; /**< This holds the number of rounds the ↩
            algorithm will use */
96      int _block_size; /**< This holds the Block size of the AES Algorithm ↩
            */
97      int _key_size; /**< The chosen key size of the algorithm */
98      byte _n; /**< This holds the initial length of key */
99      byte _b; /**< This holds the length of the expanded key */
100
101     byte _key_size_decrementer;
102     byte _m;
103
104 };
```

3.1.2 aes_implementation.cc

```
/**
 * @file aes_implementation.cc
 * @date 14/01/2018
 *
 * @brief This file contains my method definitions for my AES Implementation
 *
 * @version 0.10
 * @author Jacob Powell
 */

#include <stdexcept>
#include <cstring>
#include <iostream>
#include <iomanip>

#include "aes_implementation.h"

#define debug 1

/**< Precomputed Tables and Lookup tables are given below */

static byte sbox[256] = {
        0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
        0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
        0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
        0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
        0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
        0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
        0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
        0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
        0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
        0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
```

```
37          0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,↩
                0xE9, 0xCE, 0x55, 0x28, 0xDF,
38          0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D,↩
                0x0F, 0xB0, 0x54, 0xBB, 0x16};
39
40  static byte inverse_sbox[256] = {
41          0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3,↩
                0x9E, 0x81, 0xF3, 0xD7, 0xFB,
42          0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43,↩
                0x44, 0xC4, 0xDE, 0xE9, 0xCB,
43          0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95,↩
                0x0B, 0x42, 0xFA, 0xC3, 0x4E,
44          0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2,↩
                0x49, 0x6D, 0x8B, 0xD1, 0x25,
45          0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C,↩
                0xCC, 0x5D, 0x65, 0xB6, 0x92,
46          0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46,↩
                0x57, 0xA7, 0x8D, 0x9D, 0x84,
47          0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58,↩
                0x05, 0xB8, 0xB3, 0x45, 0x06,
48          0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD,↩
                0x03, 0x01, 0x13, 0x8A, 0x6B,
49          0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF,↩
                0xCE, 0xF0, 0xB4, 0xE6, 0x73,
50          0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37,↩
                0xE8, 0x1C, 0x75, 0xDF, 0x6E,
51          0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62,↩
                0x0E, 0xAA, 0x18, 0xBE, 0x1B,
52          0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0,↩
                0xFE, 0x78, 0xCD, 0x5A, 0xF4,
53          0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10,↩
                0x59, 0x27, 0x80, 0xEC, 0x5F,
54          0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A,↩
                0x9F, 0x93, 0xC9, 0x9C, 0xEF,
55          0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB,↩
                0x3C, 0x83, 0x53, 0x99, 0x61,
56          0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14,↩
                0x63, 0x55, 0x21, 0x0C, 0x7D};
57
58  static byte galois_mul_2[256] = {
59          0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18↩
                ,0x1a,0x1c,0x1e,
60          0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38↩
                ,0x3a,0x3c,0x3e,
61          0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58↩
                ,0x5a,0x5c,0x5e,
62          0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78↩
                ,0x7a,0x7c,0x7e,
63          0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98↩
                ,0x9a,0x9c,0x9e,
64          0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8↩
                ,0xba,0xbc,0xbe,
65          0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8↩
                ,0xda,0xdc,0xde,
```

```
66          0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8↩
                ,0xfa,0xfc,0xfe,
67          0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03↩
                ,0x01,0x07,0x05,
68          0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23↩
                ,0x21,0x27,0x25,
69          0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43↩
                ,0x41,0x47,0x45,
70          0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63↩
                ,0x61,0x67,0x65,
71          0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83↩
                ,0x81,0x87,0x85,
72          0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3↩
                ,0xa1,0xa7,0xa5,
73          0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3↩
                ,0xc1,0xc7,0xc5,
74          0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3↩
                ,0xe1,0xe7,0xe5};
75
76  static byte galois_mul_3[256] = {
77          0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14↩
                ,0x17,0x12,0x11,
78          0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24↩
                ,0x27,0x22,0x21,
79          0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74↩
                ,0x77,0x72,0x71,
80          0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44↩
                ,0x47,0x42,0x41,
81          0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4↩
                ,0xd7,0xd2,0xd1,
82          0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4↩
                ,0xe7,0xe2,0xe1,
83          0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4↩
                ,0xb7,0xb2,0xb1,
84          0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84↩
                ,0x87,0x82,0x81,
85          0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f↩
                ,0x8c,0x89,0x8a,
86          0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf↩
                ,0xbc,0xb9,0xba,
87          0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef↩
                ,0xec,0xe9,0xea,
88          0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf↩
                ,0xdc,0xd9,0xda,
89          0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f↩
                ,0x4c,0x49,0x4a,
90          0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f↩
                ,0x7c,0x79,0x7a,
91          0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f↩
                ,0x2c,0x29,0x2a,
92          0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f↩
                ,0x1c,0x19,0x1a};
93
94  static byte galois_mul_9[256] = {
```

```
95          0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c↩
                ,0x65,0x7e,0x77,
96          0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc↩
                ,0xf5,0xee,0xe7,
97          0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57↩
                ,0x5e,0x45,0x4c,
98          0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7↩
                ,0xce,0xd5,0xdc,
99          0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a↩
                ,0x13,0x08,0x01,
100         0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a↩
                ,0x83,0x98,0x91,
101         0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21↩
                ,0x28,0x33,0x3a,
102         0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1↩
                ,0xb8,0xa3,0xaa,
103         0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80↩
                ,0x89,0x92,0x9b,
104         0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10↩
                ,0x19,0x02,0x0b,
105         0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb↩
                ,0xb2,0xa9,0xa0,
106         0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b↩
                ,0x22,0x39,0x30,
107         0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6↩
                ,0xff,0xe4,0xed,
108         0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66↩
                ,0x6f,0x74,0x7d,
109         0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd↩
                ,0xc4,0xdf,0xd6,
110         0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d↩
                ,0x54,0x4f,0x46
111 };
112
113 static byte galois_mul_11[256] = {
114         0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74↩
                ,0x7f,0x62,0x69,
115         0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4↩
                ,0xcf,0xd2,0xd9,
116         0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0x0f↩
                ,0x04,0x19,0x12,
117         0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf↩
                ,0xb4,0xa9,0xa2,
118         0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82↩
                ,0x89,0x94,0x9f,
119         0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32↩
                ,0x39,0x24,0x2f,
120         0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9↩
                ,0xf2,0xef,0xe4,
121         0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49↩
                ,0x42,0x5f,0x54,
122         0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83↩
                ,0x88,0x95,0x9e,
123         0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33↩
                ,0x38,0x25,0x2e,
```

```
124         0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8↩
                ,0xf3,0xee,0xe5,
125         0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48↩
                ,0x43,0x5e,0x55,
126         0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75↩
                ,0x7e,0x63,0x68,
127         0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5↩
                ,0xce,0xd3,0xd8,
128         0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e↩
                ,0x05,0x18,0x13,
129         0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe↩
                ,0xb5,0xa8,0xa3
130 };
131
132 static byte galois_mul_13[256] = {
133         0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c↩
                ,0x51,0x46,0x4b,
134         0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c↩
                ,0x81,0x96,0x9b,
135         0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7↩
                ,0xea,0xfd,0xf0,
136         0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37↩
                ,0x3a,0x2d,0x20,
137         0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31↩
                ,0x3c,0x2b,0x26,
138         0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1↩
                ,0xec,0xfb,0xf6,
139         0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a↩
                ,0x87,0x90,0x9d,
140         0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a↩
                ,0x57,0x40,0x4d,
141         0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86↩
                ,0x8b,0x9c,0x91,
142         0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56↩
                ,0x5b,0x4c,0x41,
143         0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d↩
                ,0x30,0x27,0x2a,
144         0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed↩
                ,0xe0,0xf7,0xfa,
145         0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb↩
                ,0xe6,0xf1,0xfc,
146         0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b↩
                ,0x36,0x21,0x2c,
147         0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50↩
                ,0x5d,0x4a,0x47,
148         0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80↩
                ,0x8d,0x9a,0x97
149 };
150
151 static byte galois_mul_14[256] = {
152         0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48↩
                ,0x46,0x54,0x5a,
153         0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8↩
                ,0xa6,0xb4,0xba,
```

```
154         0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93↩
                ,0x9d,0x8f,0x81,
155         0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73↩
                ,0x7d,0x6f,0x61,
156         0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5↩
                ,0xeb,0xf9,0xf7,
157         0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05↩
                ,0x0b,0x19,0x17,
158         0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e↩
                ,0x30,0x22,0x2c,
159         0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde↩
                ,0xd0,0xc2,0xcc,
160         0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09↩
                ,0x07,0x15,0x1b,
161         0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9↩
                ,0xe7,0xf5,0xfb,
162         0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2↩
                ,0xdc,0xce,0xc0,
163         0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32↩
                ,0x3c,0x2e,0x20,
164         0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4↩
                ,0xaa,0xb8,0xb6,
165         0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44↩
                ,0x4a,0x58,0x56,
166         0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f↩
                ,0x71,0x63,0x6d,
167         0xd7,0xd9,0xcb,0xc5,0xef,0xe1,0xf3,0xfd,0xa7,0xa9,0xbb,0xb5,0x9f↩
                ,0x91,0x83,0x8d
168 };
169
170 static byte rcon[256] = {
171         0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,↩
                0x6c, 0xd8, 0xab, 0x4d, 0x9a,
172         0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,↩
                0xfa, 0xef, 0xc5, 0x91, 0x39,
173         0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,↩
                0x66, 0xcc, 0x83, 0x1d, 0x3a,
174         0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,↩
                0x80, 0x1b, 0x36, 0x6c, 0xd8,
175         0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,↩
                0xd4, 0xb3, 0x7d, 0xfa, 0xef,
176         0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,↩
                0x4a, 0x94, 0x33, 0x66, 0xcc,
177         0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08,↩
                0x10, 0x20, 0x40, 0x80, 0x1b,
178         0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,↩
                0x97, 0x35, 0x6a, 0xd4, 0xb3,
179         0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,↩
                0xc2, 0x9f, 0x25, 0x4a, 0x94,
180         0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,↩
                0x02, 0x04, 0x08, 0x10, 0x20,
181         0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,↩
                0xbc, 0x63, 0xc6, 0x97, 0x35,
182         0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4,↩
                0xd3, 0xbd, 0x61, 0xc2, 0x9f,
```

```
183         0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,↩
                0xcb, 0x8d, 0x01, 0x02, 0x04,
184         0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,↩
                0x9a, 0x2f, 0x5e, 0xbc, 0x63,
185         0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,↩
                0x39, 0x72, 0xe4, 0xd3, 0xbd,
186         0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,↩
                0x3a, 0x74, 0xe8, 0xcb, 0x8d
187 };
188
189
190 AESImplementation::AESImplementation(AESKeyLengths keyLength) : _n(0),
191                                                                 _b(0),
192                                                                 _number_of_rounds↩
                                                                        (0),
193                                                                 _block_size↩
                                                                        (0){
194     switch(keyLength){
195         case AES128:
196             this->_block_size = 16;
197             this->_key_size = 128;
198             this->_number_of_rounds = 10;
199             this->_n = 16;
200             this->_b = 176;
201             this->_m = 0;
202             this->_key_size_decrementer = 16;
203             break;
204         case AES192:
205             this->_block_size = 16;
206             this->_key_size = 192;
207             this->_number_of_rounds = 12;
208             this->_n = 24;
209             this->_b = 208;
210             this->_m = 2;
211             this->_key_size_decrementer = 24;
212             break;
213         case AES256:
214             this->_block_size = 16;
215             this->_key_size = 256;
216             this->_number_of_rounds = 14;
217             this->_n = 32;
218             this->_b = 240;
219             this->_m = 3;
220             this->_key_size_decrementer = 32;
221             break;
222         default:
223             throw std::runtime_error("Invalid key length");
224     }
225 }
226
227 void AESImplementation::encrypt(const byte input[], byte output[], const ↩
      byte key[]) {
228
229     std::cout << "Encryption Process Started" << std::endl;
230
```

```
231     for(byte i = 0; i < 4; i++){
232         this->_state[i][0] = input[i];
233         this->_state[i][1] = input[i + 4];
234         this->_state[i][2] = input[i + 8];
235         this->_state[i][3] = input[i + 12];
236     }
237 
238     this->outputState("Current State");
239 
240     byte expanded_key[this->_b];
241     this->KeyExpansion(key, expanded_key);
242 
243     for(byte i = 0; i < 4; i++){
244         this->_round_key[i][0] = expanded_key[i];
245         this->_round_key[i][1] = expanded_key[i + 4];
246         this->_round_key[i][2] = expanded_key[i + 8];
247         this->_round_key[i][3] = expanded_key[i + 12];
248     }
249 
250     this->outputRoundKey();
251 
252     this->AddRoundKey(this->_state, this->_round_key);
253 
254     this->outputState("Current State");
255 
256     for(byte current_round = 1; current_round <= this->_number_of_rounds;↩
        current_round++){
257         std::cout << "Start of Round " << unsigned(current_round) << std↩
            ::endl;
258         for(byte i = 0; i < 4; i++)
259             for(byte j = 0; j < 4; j++) {
260                 this->_round_key[i][j] = expanded_key[(current_round * ↩
                    this->_block_size) + (j * 4) + i];
261 
262             }
263 
264         this->SubBytes(this->_state);
265         this->ShiftRows(this->_state);
266         if(current_round != this->_number_of_rounds) {
267             this->MixColumns(this->_state);
268         }
269         this->AddRoundKey(this->_state, this->_round_key);
270         this->outputRoundKey();
271         std::cout << "End of Round " << unsigned(current_round) << std::↩
            endl;
272     }
273 
274     for(byte i = 0; i < 4; i++){
275         for(byte j = 0; j < 4; j++){
276             output[(j * 4) + i] = this->_state[i][j];
277         }
278     }
279     std::cout << "Encryption Process Ended" << std::endl;
280 }
281 
```

```
282 void AESImplementation::decrypt(const byte input[], byte output[], const ↩
    byte key[]) {
283
284     std::cout << "Decryption Process Started" << std::endl;
285
286     for(byte i = 0; i < 4; i++){
287         this->_state[i][0] = input[i];
288         this->_state[i][1] = input[i + 4];
289         this->_state[i][2] = input[i + 8];
290         this->_state[i][3] = input[i + 12];
291     }
292
293     this->outputState("Current State");
294
295     byte expanded_key[this->_b];
296     this->KeyExpansion(key, expanded_key);
297
298
299     for(byte i = 0; i < 4; i++){
300         this->_round_key[i][0] = expanded_key[this->_b   (16   i)];
301         this->_round_key[i][1] = expanded_key[this->_b   (12   i)];
302         this->_round_key[i][2] = expanded_key[this->_b   (8   i)];
303         this->_round_key[i][3] = expanded_key[this->_b   (4   i)];
304     }
305
306     this->outputRoundKey();
307
308     this->AddRoundKey(this->_state, this->_round_key);
309
310     this->outputState("Start: ");
311
312     for(byte current_round = (this->_number_of_rounds   1); current_round↩
        > 0; current_round  ){
313         std::cout << "Start of Round " << unsigned(current_round) << std↩
            ::endl;
314         this->outputState("Current State: ");
315         for(byte i = 0; i < 4; i++) {
316             for (byte j = 0; j < 4; j++) {
317                 this->_round_key[i][j] = expanded_key[(current_round * ↩
                    this->_block_size) + (j * 4) + i];
318             }
319         }
320
321         this->InverseSubBytes(this->_state);
322         this->InverseShiftRows(this->_state);
323
324         this->outputRoundKey();
325         this->AddRoundKey(this->_state, this->_round_key);
326
327         if(current_round != 0){
328             this->InverseMixColumns(this->_state);
329         }
330
331         std::cout << "End of Round " << unsigned(current_round) << std::↩
            endl;
```

```
332     }
333
334     this->outputState("End of Dec");
335
336     for(byte i = 0; i < 4; i++){
337         for(byte j = 0; j < 4; j++){
338             output[(j * 4) + i] = this->_state[i][j];
339         }
340     }
341
342     std::cout << "Decryption Process Ended" << std::endl;
343 }
344
345
346 void AESImplementation::KeyExpansion(const byte key[], byte expandedKey[]) const {
347
348 #if defined(debug)
349     std::cout << "Key Expansion Start" << std::endl;
350 #endif
351     memset(expandedKey, 0, this->_b);
352     memcpy(expandedKey, key, this->_n);
353
354 #if defined(debug)
355     std::cout << "W[0 3]:";
356     for(byte i = 0; i < this->_n; i++){
357         std::cout << std::hex << std::setfill('0') << std::setw(2) << unsigned(expandedKey[i]);
358     }
359     std::cout << std::endl;
360 #endif
361
362     byte keySizeIterator = 0;
363     keySizeIterator += this->_n;
364
365     byte t[4];
366     byte u[4];
367
368     for(byte rcon = 1; keySizeIterator < this->_b; rcon++){
369
370         memcpy(t, expandedKey + (keySizeIterator - 4), 4);
371
372 #if defined(debug)
373         std::cout << "PRE 4: ";
374         for(byte i = 0; i < 4; i++)
375             std::cout << std::hex << std::setfill('0') << std::setw(2) << unsigned(t[i]);
376         std::cout << std::endl;
377 #endif
378
379         this->KeyExpansionCore(rcon, t, u);
380         memcpy(t, expandedKey + (keySizeIterator - this->_n), 4 * sizeof(byte));
381
382 #if defined(debug)
```

```
383         std::cout << "W[i   4]: ";
384         for(byte i = 0; i < 4; i++)
385             std::cout << std::hex << std::setfill('0') << std::setw(2) <<↩
                unsigned(u[i]);
386         std::cout << std::endl;
387 #endif
388 
389         for(byte i = 0; i < 4; i++)
390             t[i] ^= u[i];
391 
392 #if defined(debug)
393         std::cout << "W[i]: ";
394         for(byte i = 0; i < 4; i++)
395             std::cout << std::hex << std::setfill('0') << std::setw(2) <<↩
                unsigned(t[i]);
396         std::cout << std::endl;
397 #endif
398 
399         memcpy(expandedKey + keySizeIterator, t, 4 * sizeof(byte));
400         keySizeIterator += 4;
401 
402         for(byte i = 0; i < 3 && (keySizeIterator < this->_b); i++){
403             memcpy(t, expandedKey + (keySizeIterator   4), 4 * sizeof(↩
                byte));
404 
405             std::cout << "temp: ";
406             for (byte j = 0; j < 4; j++)
407             {
408                 expandedKey[keySizeIterator + j] = t[j] ^ expandedKey[↩
                    keySizeIterator   this->_n + j];
409                 std::cout << (unsigned)expandedKey[keySizeIterator + j];
410             }
411             std::cout << std::endl;
412 
413             keySizeIterator += 4;
414         }
415 
416         if((this->_key_size == 256) && (keySizeIterator < this->_b)){
417 
418             memcpy(t, expandedKey + (keySizeIterator   4), 4 * sizeof(↩
                byte));
419             for(byte i = 0; i < 4; i++){
420                 t[i] = sbox[t[i]];
421             }
422 
423             for(byte i = 0; i < 4; i++){
424                 expandedKey[keySizeIterator + i] = t[i] ^ expandedKey[↩
                    keySizeIterator   this->_n + i];
425                 t[i] ^= expandedKey[keySizeIterator   this->_n   4 + i];
426             }
427             //memcpy(expandedKey + (keySizeIterator   3   1), t, 4 * ↩
                sizeof(byte));
428             keySizeIterator += 4;
429         }
430 
```

```
431         for(byte i = 0; (i < this->_m) && (keySizeIterator < this->_b); i++){
432             memcpy(t, expandedKey + (keySizeIterator    4), 4 * sizeof(byte));
433 
434             std::cout << "temp: ";
435             for (byte j = 0; j < 4; j++)
436             {
437                 expandedKey[keySizeIterator + j] = t[j] ^ expandedKey[keySizeIterator    this->_n + j];
438                 std::cout << (unsigned)expandedKey[keySizeIterator + j];
439             }
440             std::cout << std::endl;
441 
442             keySizeIterator += 4;
443         }
444 
445     }
446 
447 #if defined(debug)
448     int count = 0;
449     int roundNumber = 1;
450     std::cout << "Expanded Key: ";
451     for(byte i = 0; i < this->_b; i++){
452         if(count % 16 == 0)
453             std::cout << std::endl;
454 
455         std::cout << std::hex << std::setfill('0') << std::setw(2) << unsigned(expandedKey[i]);
456         count++;
457         roundNumber++;
458     }
459 
460     std::cout << std::endl;
461     std::cout << "Key Expansion End" << std::endl;
462 #endif
463 }
464 
465 void AESImplementation::KeyExpansionCore(byte roundNumber, const byte keyIn[4], byte keyOut[4]){
466 
467     //std::cout << "Key Expansion Core Start" << std::endl;
468 
469     //std::cout << "KEY OUT: ";
470 
471     /*for(byte i = 0; i < 4; i++){
472         std::cout << std::hex << std::setfill('0') << std::setw(2) << unsigned(keyOut[i]);
473     }
474     std::cout << std::endl;*/
475 
476     //std::cout << "KEY IN: ";
477 
478     /*for(byte i = 0; i < 4; i++){
```

```
479         std::cout << std::hex << std::setfill('0') << std::setw(2) << ↩
                unsigned(keyIn[i]);
480     }
481     std::cout << std::endl;*/
482 
483     memcpy(keyOut, keyIn, 4);
484 
485     /**
486      * Rotation Step
487      */
488     byte temp = keyOut[0];
489     for(byte i = 0; i < 3; i++){
490         keyOut[i] = keyOut[i+1];
491     }
492     keyOut[3] = temp;
493 
494     /*std::cout << "After Rotation: ";
495 
496     for(byte i = 0; i < 4; i++){
497         std::cout << std::hex << std::setfill('0') << std::setw(2) << ↩
                unsigned(keyOut[i]);
498     }
499     std::cout << std::endl;*/
500 
501     /**
502      * Substitution Step
503      */
504     for(byte i = 0; i < 4; i++){
505         keyOut[i] = sbox[keyOut[i]];
506     }
507 
508     /*std::cout << "After Substitution: ";
509     for(byte i = 0; i < 4; i++){
510         std::cout << std::hex << std::setfill('0') << std::setw(2) << ↩
                unsigned(keyOut[i]);
511     }
512     std::cout << std::endl;*/
513 
514 
515     keyOut[0] = keyOut[0] ^ rcon[roundNumber];
516 
517     /*std::cout << "After XOR With RCON: ";
518     for(byte i = 0; i < 4; i++){
519         std::cout << std::hex << std::setfill('0') << std::setw(2) << ↩
                unsigned(keyOut[i]);
520     }
521     std::cout << std::endl;*/
522 
523     //std::cout << "Key Expansion Core End" << std::endl;
524 
525 }
526 
527 void AESImplementation::AddRoundKey(byte state[4][4], byte roundKey↩
    [4][4]) {
528     for(byte i = 0; i < 4; i++){
```

```
529         auto* key = reinterpret_cast<uint32_t *>(roundKey[i]);
530         auto* row_state = reinterpret_cast<uint32_t *>(state[i]);
531         *row_state ^= *key;
532     }
533 }
534
535 void AESImplementation::SubBytes(byte state[4][4]) {
536     for(byte i = 0; i < 4; i++)
537         for(byte j = 0; j < 4; j++)
538             state[i][j] = sbox[state[i][j]];
539     this->outputState("After SubBytes: ");
540
541 }
542
543 void AESImplementation::ShiftRows(byte state[4][4]) {
544     for(byte i = 1; i < 4; i++){
545         byte row[4];
546         for(byte j = 0; j < 4; j++)
547             row[j] = state[i][j];
548         for(byte j = 0; j < 4; j++)
549             state[i][j] = row[(i + j) % 4];
550     }
551     this->outputState("After ShiftRows: ");
552 }
553
554 void AESImplementation::MixColumns(byte state[4][4]) {
555
556     for(byte i = 0; i < 4; i++){
557         byte column[4] = {state[0][i], state[1][i], state[2][i], state[3][i],};
558
559         state[0][i] = galois_mul_2[column[0]] ^ galois_mul_3[column[1]] ^ column[2] ^ column[3];
560         state[1][i] = column[0] ^ galois_mul_2[column[1]] ^ galois_mul_3[column[2]] ^ column[3];
561         state[2][i] = column[0] ^ column[1] ^ galois_mul_2[column[2]] ^ galois_mul_3[column[3]];
562         state[3][i] = galois_mul_3[column[0]] ^ column[1] ^ column[2] ^ galois_mul_2[column[3]];
563     }
564     this->outputState("After MixColumns: ");
565 }
566
567
568
569 void AESImplementation::InverseSubBytes(byte state[4][4]) {
570     for(byte i = 0; i < 4; i++)
571         for(byte j = 0; j < 4; j++)
572             state[i][j] = inverse_sbox[state[i][j]];
573     this->outputState("After InverseSubBytes: ");
574 }
575
576 void AESImplementation::InverseShiftRows(byte state[4][4]) {
577     for(byte i = 1; i < 4; i++){
578         byte row[4];
```

```
579         for(byte j = 0; j < 4; j++)
580             row[j] = state[i][j];
581         for(int k = 3; k >= 0; k  ){
582             state[i][k] = row[(k + (4   i)) % 4];
583         }
584     }
585     this >outputState("After InverseShiftRows: ");
586 }
587
588 void AESImplementation::InverseMixColumns(byte state[4][4]) {
589     for(byte i = 0; i < 4; i++){
590         byte column[4] = {state[0][i], state[1][i], state[2][i], state↩
                [3][i]};
591
592         state[0][i] = galois_mul_14[column[0]] ^ galois_mul_11[column[1]]↩
                ^ galois_mul_13[column[2]] ^ galois_mul_9[column[3]];
593         state[1][i] = galois_mul_9[column[0]] ^ galois_mul_14[column[1]] ↩
                ^ galois_mul_11[column[2]] ^ galois_mul_13[column[3]];
594         state[2][i] = galois_mul_13[column[0]] ^ galois_mul_9[column[1]] ↩
                ^ galois_mul_14[column[2]] ^ galois_mul_11[column[3]];
595         state[3][i] = galois_mul_11[column[0]] ^ galois_mul_13[column[1]]↩
                ^ galois_mul_9[column[2]] ^ galois_mul_14[column[3]];
596     }
597     this >outputState("After InverseMixColumns: ");
598 }
599
600 void AESImplementation::outputState(std::string prefix) {
601 #if defined(debug)
602     std::cout << prefix << std::endl;
603     for(byte i = 0; i < 4; i++){
604         for(byte j = 0; j < 4; j++)
605             std::cout << std::hex << std::setfill('0') << std::setw(2) <<↩
                    unsigned(this >_state[j][i]);
606     }
607     std::cout << std::endl;
608 #endif
609 }
610
611 void AESImplementation::outputRoundKey() {
612 #if defined(debug)
613     std::cout << "Current Round Key" << std::endl;
614     for(byte i = 0; i < 4; i++){
615         for(byte j = 0; j < 4; j++)
616             std::cout << std::hex << std::setfill('0') << std::setw(2) <<↩
                    unsigned(this >_round_key[j][i]);
617     }
618     std::cout << std::endl;
619 #endif
620 }
```

3.2 DES Implementation

3.3 Web Application Implementation

Chapter 4

Testing

Chapter 5

Evaluation

List of Tables

List of Figures