Creating a new encryption algorithm using the structure of existing algorithms to increase computational complexity.

Joshua Cartwright

BSc (Hons) Cyber Security

The qualification is awarded by Staffordshire University and this report is submitted in partial fulfilment of the BSc (Hons) Cyber Security course.

Supervisor: Mark Stanier

Assessor: Derek A Colley

Submission Date: 10/05/2021

Contents

Introduction

This project aims to improve upon Advanced Encryption Standard (AES) which has not been updated in 20 years. The improvements to be added should be designed to increase computational complexity and make the algorithm harder to brute force. To achieve this AES will be rebuilt to research how it works and to give a starting point for the project. From there functions of other block cyphers that could be merged with AES to improve the computational complexity will be investigated. Research into other methods that might help improve the computational complexity will be conducted, and the previously mentioned methods will be coded into AES to create a small command-line interface. One of the main pieces of work that will inform and influence this project is the official AES documentation. (*FIPS PUBS, 2001*)

AES encryption is currently used in many ways, for example, wireless security, processor security, file encryption and Secure Sockets Layer/Transport Layer Security (SSL/TLS). Although AES has not been brute forced or otherwise

compromised it has not been updated since its release in 2001. AES has proven to be very efficient and effective, and given the correct key, adds little noticeable difference in overhead for any prosses in which it is utilized. AES is a fast and highly secure form of encryption that is a popular choice amongst businesses and governments. However, its level of effectiveness varies depending on its implementation unlike the brute force attacks, effective attacks are typically on the implementation of AES as opposed to the algorithm itself, instead of brute forcing the algorithm attackers can target the users. The way in which this is done is through Phishing Attacks. These attacks are known as and considered Side Channel Attacks; this is due to where within the algorithm the attacks are being carried out (a side channel attack is when an attacker monitors the user's pc to gain access to their encryption key). For example, not through the focal point of the security implementation but other aspects of the entire process. (*Thomas R, 2020*)

Project Aims

In this project the aim is to create a better version of AES, this better version should be even more secure and resistant to brute force attack. To achieve this, AES will be coded exactly as it is within the 2001 standard (*FIPS PUBS, 2001*). This will give a greater understanding of how AES works and will give a baseline to work off of. From this baseline, other encryption algorithms will be researched to find techniques that could be implemented to increase security. The maximum key size will be expanded from 256-bit to 512-bit or higher this will make the key harder to brute force. Towards the end of this project, the aim is to have a fully functional console application that can encrypt files from the computer.

Ethical Considerations

This project will abide by the British Computer Society Code of Conduct. Abiding by these regulations is fairly easy as the project does not include any involvement with others outside of university and examiners. The algorithm will be offline and will only store data temporarily in ram and so cannot be used in a way that would breach someone's privacy and cannot cause harm or endanger someone's wellbeing. Because the software is purely mathematical in nature it cannot be used to discriminate on any grounds. Under the section of professional competence and integrity I the author cannot claim and level of competence I do not possess. This project is undertaken as a university student studying cybersecurity and I have undertaken a project in the past at an A-Level creating an improved version of the ADFGVX cypher. This project is also undertaken within the scope of UK law. This project will not bring the computing sector into disrepute. (*BCS, no date*)

Previous Experience

While in college we did a class on cyphers including the infamous Ceaser cypher, during my time there we were usually given task sheets to complete before we could leave for lunch these had hundreds of questions around the given topic. So that we could leave for lunch on time we created small programs in C# to solve the entire sheet in seconds (including time to program this was normally quicker than solving the sheet by hand). After I created a Ceaser cypher program some of my friends were discussing getting an unblocked messaging service on the college computers. I created a version of the ADFGVX cypher with a shifting key that changes the location of the cells. This was then incorporated into an online Webservice with my algorithm encrypting all text stored. Even though the software did not last long before the admins asked me to remove it the creation of this software gave me valuable experience in both programming and cryptography.

Previous encrypted chat room project.

Risk Assessment

Project Management

At the beginning of this project, there were a few delays and problems going into the project. The project was started with very little understanding of how AES worked It took months to work out all of the different functions that make AES work. Within the first month, I upgraded my laptop and for about a week I did not have a computer and with Covid-19 happening outside I could not work on the project or attend lectures, tutorials, or meetings for this project. From November I had to stop working on the project as I had assignments due for other modules, during this time I ran into mental health issues that slowed progress on all modules. after Christmas, work resumed to get the project back on track coding AES as set in the 2001 standard was completed. The projected then moved on to the next stage researching methods to improve the algorithm. Within 2 weeks the coding part of the project was completed.

Supervisory Meetings

Every Monday at 1 pm there is a meeting with the project supervisor. In these meeting changes that have been made to the project over the last week are discussed along with the goals for the next week to stay on track.

Meetings Log

Date	Attendance	Notes
19 October 2020	TRUE	Discussed Project Aims and Previous Experience
26 October 2020	TRUE	Discussed Project Timeline and Deadlines
02 November 2020	TRUE	Discussed AES Improvements and Looked At Finished Gant Chart For Timeline
09 November 2020	Postponed to 11/11/2020	Unable to attend postponed
11 November 2020	TRUE	Discussed Code (Start function) and next steps
23 November 2020	FALSE	Unable to attend
29 November 2020	TRUE	Discussed Code (SubBytes& ShiftRows)
05 December 2020	TRUE	Discussed Code (Mix Columns & AddRoundKey)
14 December 2020	TRUE	Discussed Improvements To AES

11 January 2021	TRUE	Mid-Point Review
18 January 2021	FALSE	Unable to attend
25 January 2021	TRUE	Discussed Code (Key Expand)
01 February 2021	TRUE	Discussed Code (Key Expand 192 & 256 Bit)
08 February 2021	TRUE	Discussed Code (Key Expand 384 bit & 512 Bit)
15 February 2021	TRUE	Discussed Code (Interlaced Key)
22 February 2021	TRUE	Discussed Code Testing and Final Edits
08 March 2021	TRUE	Discussed Writeup
15 March 2021	TRUE	Discussed Writeup
22 March 2021	TRUE	Discussed Writeup
14 April 2021	FALSE	Unable to attend
21 April 2021	FALSE	Unable to attend
28 April 2021	FALSE	Unable to attend
05 May 2021	FALSE	Unable to attend

Methodology

The methodology chosen for this project is rapid application development as it allows quick alterations to the design while prototyping the code. "*Rapid Application Development (RAD) is a form of agile software development methodology that prioritizes rapid prototype releases and iterations. Unlike the Waterfall method, RAD emphasizes the use of software and user feedback over strict planning and requirements recording."* (Singh A, 2019). This is an asset in this project as the functions of AES are being researched as they are coded this gives a greater understanding of how the algorithm works. To minimise the risk of failure the code will be broken down into small manageable functions so that they can be worked on separately, this means that tested functions can be bug free and will help avoid confusion while programming.

Research

Encryption Algorithms

An encryption algorithm is a way to conceal the true meaning of any type of data. Encryption stems from the ancient Greeks around 400BC and was used to relay secret messages between military commanders. For instance, the infamous Ceasar cypher shifts letters in the alphabet so that an A turns into a D and a B into an E this pattern continues for the full alphabet so that a message like "Attack at Dawn" turns into "Dwwdfn Dw Gdzq". Modern encryption is more complex and involves mathematical formulas that are easy to compute one way but not the other for example a xor operation. (*GeeksforGeeks, 2021*)

	0	1
0	0	1
1	1	0

Xor operation table.

A xor takes two numbers and only outputs a 1 if the inputs are different. This is used in encryption because even if the message reader knows the output, they cannot get either of the inputs unless they already know one. For example, if someone intercepted the binary number 101 from a hidden message using xor, the actual message could be any number. here is a table of all the possibilities.

Encrypted Message	Possible Key	Xor Result
101	000	101
101	001	100
101	010	111
101	011	110
101	100	001
101	101	000
101	110	011
101	111	010

As demonstrated above, there is a list of every possible combination and the message reader has to pick the one that looks correct, this is called a brute force attack. Now if they know the key "001" they can instantly work out that 101 xor 001 = 100 and have decrypted the message revealing 4 as the secret number. (*PCMag, no date*)

The type of modern encryption that will be focused on in this project is called substitution permutation networks. These work by substituting data for other data in a substitution box. This is what a substitution box looks like.

	A	B
A	AB	BB
B	AA	BA

This is used to obscure values, for example, say the message writer has "AA" as their hidden message it is turned into "AB" to create encryption. This along with xoring with a key and functions to move cells around creates confusion and diffusion. These are techniques for hiding the relation between plaintext and ciphertext (encrypted text) and hiding the relation between ciphertext and the key. (*Sage, no date*)

Say the message writer uses all these techniques for this example A will be treated as a 0 in binary and B as 1, the message writers plaintext will be "AA" and there key will be "BB" first they will run substitution on the plaintext "AA" = "AB" then they will swap the two letters "BA" = "AB" now they will xor it with the key "01" xor "11" = "10" = "BA" this is there cyphertext and is encrypted. Now for the permutation, this bit is easy as they just loop what is shown above but with a new key each round, in modern encryption many keys are made from the original key that the user inputs these are called sub-keys or round keys. The focus for this project is on AES which is a substitution permutation network algorithm. (*Educative, no date*)

AES

AES otherwise known as Rijndael consists of several main functions which it loops to encrypt data. These are as follows: SubBytes, ShiftRows, MixColums, AddRoundKey and KeyExpansion. On the first round of encryption key expansion is used to create an array of subkeys to be used in the AddRoundKey function. Then the AddRoundKey function is run. After this, a loop runs SubBytes, ShiftRows, MixColums and AddRoundKey in that order. In the final round, MixColumns is not run. AES has never been cracked yet and is safe against any brute force attacks (Mohit Arora, 2012) one of the main drawbacks of AES is that all blocks are encrypted with the same key, for example if you encrypt a file with a size larger than 256 bit all 128-bit blocks of the file are encrypted with the same set of round keys. This could be solved by using an equal size key to the data you are trying to encrypt and use each 128-bit block to make unique keys for each block of encryption or you could expand a small key until enough key blocks are made. Neither of the previously mentioned methods is included in the standard as the standard itself only looks at encrypting 1 block of data with a variable size key, this means both of these methods are up to the developer adding changes before encryption is run. (*FIPS PUBS, 2001*)

SubBytes

SubBytes is the function that adds substitution to AES. Substitution adds diffusion to the algorithm. Diffusion means that the algorithm is hiding the link between the plaintext and cyphertext. For example, if the location of letters in a word are changed it can easily be figured out e.g. olhle = hello but if every character is changed to the character after it like in a Ceaser cypher: olhle > pmimf, it cannot easily be turned back into hello. It does this by using a substitution box, this sbox is predefined in the standard. The function works on a cell-by-cell basis. It takes the current cell and uses its hexadecimal characters as gird coordinates to find a value in the sbox. For example, A4 would take the 11th cell down and the 5th cell across in the sbox getting the value 49 (in hexadecimal format) (*FIPS PUBS, 2001, p.16 fig. 7*).

ShiftRows

The ShiftRows function is a function that adds confusion to AES. Confusion is another way of hiding the link between any bit in the cyphertext and the key. This works by moving the values around so that a cell we xored a key with last round will be in a different location in the array this round. Shift rows works on a row-by-row basis and are the simplest of the functions in AES. The first row in the block is unaffected and remains the same, however, in the second row all cells are shifted left 1 this wraps around so the leftmost cell becomes the rightmost cell. On the third row, the cells are shifted 2 to the left and for the fourth row, the cells are moved 3 to the left. (*FIPS PUBS, 2001, p.17 fig. 8*).

Mixcolumns

The Mixcolumns Function is a function that adds confusion and diffusion to AES. When looking at the documentation for the function it was very difficult to understand the maths behind the function so here is an explanation as to how it works without the maths. The Mixcolums function works on a cell-by-cell basis. After selecting a cell, it grabs the entire column the cell is in, it then uses a grid named the mix-grid it looks like this:

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

There is also an inverse mix-grid for decrypting. the function then uses the row of the current cell and gets the entire row of the same row on the mix-grid. Each cell in the column starting from the top is then multiplied with the mix-grid starting from the left in a Galois field, according to research is multiplication that loops around once the value exceeds 255, due to a lack of understanding the code will use lookup tables instead. After each cell has been multiplied with its corresponding cell the 4 values are xored together to get the final value for the original cell, this is added to a temporary block to prevent it from affecting the next cell. (*FIPS PUBS, 2001*)

AddRoundKey

The add round key function is the part where the key comes into play. Each cell is xored with the corresponding cell in the current round key (round keys are generated in the initial KeyExpantion). This makes it so that the ciphertext cannot be decrypted without the key as there is no way to know what the block looked like before being xored. This works because the xored value could have any bit changed and there's no way of finding out without knowing the key, for example, 1001 is the result of two xored values but unless we know one of those values (key and plaintext) we cannot get the original number, but if the key is known to be 0101 xor can be used to get it back to the original e.g. ???? xor 0101 = 1001, ???? = 1001 xor 0101 = 0100. (*FIPS PUBS, 2001*)

KeyExpantion

KeyExpantion is the longest and most complex function within AES, it is responsible for turning the user inputted key into a bunch of subkeys also known as round keys. The key expansion has many rounds and works on a column-by-column basis. To start the last column of the previous key (on the first round the last column of the user inputted key) is put into a function called G (covered after this). The column returned from G is then xored with the first column of

the previous key, this column will be the first column of the new subkey. The second column is made by xoring the column we just made with the second column of the previous key. The third is made with the second column and the third from the previous key and the fourth column is made from the third column and the fourth from the previous key. For key sizes above 128-bit, the blocks are just made bigger, 192-bit is a 6x4 block and 256-bit is an 8x4 block, for these blocks the cycle of xoring the previous column and the same column on the previous block is continued. For 256-bit key expansion, the 4th column goes through a function called Subword after it is created. (*FIPS PUBS, 2001*)

G function

The G function is separated into 3 smaller functions that run a given column in this order: RotWord, SubWord and XorRcon. (*FIPS PUBS, 2001*)

RotWord

This function is very simple and just takes the column and returns it with all cells shifted left 1 space e.g., 1,2,3,4 = 2,3,4,1. (*FIPS PUBS, 2001*)

SubWord

This function is a small-scale version of the subBytes function earlier on it takes each cell in the column and uses its value in hexadecimal to determine its coordinates in a substitution box then uses that as the new value and returns the column. (*FIPS PUBS, 2001*)

XorRcon

This function xors the first cell with a hexadecimal value in a lookup table. The cell in the lookup table is determined by what round of key expantion the function is on. These are the values:
"01","02","04","08","10","20","40","80","1B","36". (*FIPS PUBS, 2001*)

Twofish

The Twofish cypher has an interesting approach, it uses two key-dependent s-boxes. This means that the s-boxes used by the sub byte's function are generated by the key scheduler. Theoretically key-dependent s-boxes should increase the security of AES. This should add diffusion as the link between the cyphertext and the key as the key is used in more than one place. However according to Sean Murphy, 2002 "the use of key-dependent S-boxes can potentially improve the range of options available to the attacker". This could be an interesting approach if the project has any spare time. (*Stoianov N, 2011*)

Blowfish

Research was undertaking to look into Blowfish which seems similar to Twofish but uses four s-boxes and has a block size of 64-bit." *Blowfish is a symmetric encryption algorithm, meaning that it uses the same secret key to both encrypt and decrypt messages. Blowfish is also a block cipher, meaning that it divides a message up into fixed length blocks during encryption and decryption. The block length for Blowfish is 64 bits; messages that aren't a multiple of eight bytes in size must be padded."* (Gatliff B, 2003). However, Twofish has better security than Blowfish so it was decided not to continue the researching into this algorithm.

Key sizes above 256-bit

When looking online a forum was found discussing key sizes above 256-bit, users on the forum said the reality is that the software was using 2 256-bit keys to emulate 512-bit. From this a similar design was created, 2 keys will be used for encrypting and both will be expanded. To create the final key round keys will be drawn from the top of the stack from key 1 and then again using a round key from key 2 from there round keys will be drawn of each stack until none are left this will be called an interlaced key.

Asymmetric encryption

AES is a symmetric encryption algorithm; this means that there is one private key, if an attacker gets this private key all data sent from the target with the intercepted key can be decrypted. With asymmetric encryption there are 2 keys; a

private key like in symmetric encryption that can decrypt data, as well as a public key that can encrypt data but cannot be used to decrypt it. "*The public key is also called asymmetric cryptography.*" (Parahar M, 2020). An example of an algorithm that uses asymmetric encryption is Digital Signature Algorithm (DSA), this is a type of public-key encryption algorithm, and it is used to generate an electronic signature. (*Pedamkar P, no date*)

When two users communicate using asymmetric encryption, they both generate a public and private key then send the other user their public key. Both users can send information to the other by using their partners public key to encrypt the data, the user on the other end can then use their private key to decrypt the received data. Any attacker intercepting the 2 users' data cannot decrypt this data as neither private key has been sent. According to Ryan Yackel, 2020 symmetric encryption is much faster than asymmetric, this is largely in part to asymmetric encryption's large key size and because only 1 key is used for the algorithm. On the other hand, asymmetric has better security than symmetric as asymmetric encryption has a key that can be shared with others without the risk that attackers intercept the key and decrypt the data.

Symmetric encryption is often chosen over asymmetric encryption for data storage as there is only 1 user and so there is no need to share a key. Ryan Yackel, 2020 describes some of the common cases symmetric and asymmetric encryption is used for, asymmetric algorithms are commonly used for digital signatures, blockchain and public key infrastructure, of which none of these categories have a place in this project. Symmetric algorithms are commonly used for banking and data storage, my project will only be looking to encrypt store and decrypt data for this reason symmetric encryption is a better fit for this project.

Blockchain

Block chain is a method of securing data using peer-to-peer networks and hashes for authentication. Blockchain works by sorting data into blocks, each block has 3 critical parts: data, hash and previous block hash. The data section of a block contains all the data that needs to be sent or stored. The hash section is a hash of the data within the block and the previous block hash contains the hash of the previous block. The way blockchain works is that each new block of data contains the hash of the old block, this way whenever a previous block is changed its hash will change and all the blocks following it become invalid as their previous hash value will not match the previous expected hash. This can be used to detect if someone has tried to intercept and change a blocks data. Because the blocks are linked together like this, they are called a block-chain. After a block has been generated, they are usually distributed amongst a peer-to-peer network, this way if one user tries to modify a block chain and changes all following blocks their node will be considered malicious as the other nodes will check there copy of the block and come to a consensus between them that the block sent is invalid. (Simply Explained, 2017)

To successfully attack a block chain over 50% of the network's nodes must be attacked and all blocks past the first change point must be re-calculated. Blockchain nodes must also complete a complex algorithm or wait a period of time before appending blocks, this helps prevent malicious nodes. Once a node has helped verify other blocks and its own blocks have been verified the nodes trust goes up within the network as this goes up less of its blocks will need verifying saving time and resources for the node. As blockchain is a technology that requires a network of nodes it cannot be using in this project as the intention is to encrypt and decrypt data on a single computer.

However, the algorithm designed within this project could be used pre-blockchain to encrypt data before they are put into blocks, this can be used to hide the contents of the blockchain from the peer-to-peer network. Data from my encryption could also use a modified blockchain without the peer-to-peer network as a validation method while the data is stored, this could use the hash and previous block hash to confirm or deny weather data stored in the encrypted format (128-bit blocks) had been tampered with however, without the peer-to-peer network an attacker could simply re-write all of the hash values for the cyphertext. (*Conway L, 2020*)

Triple Data Encryption Standard

Triple Data Encryption Algorithm (3DES) is a symmetric key block cypher. 3DES has a block size of 64-bit. 3DES was the main encryption algorithm used before AES replaced it (Townsend Security, 2020) and is still broadly used in. It is up for retirement soon as "*the collision attack on TDEA represents a serious security vulnerability for many common uses of these protocols*" (NIST, 2017). 3DES works based on the Data Encryption Standard (DES) however uses 3 keys and performs the following: encrypts data using DES with the 1st key, decrypts the cyphertext using DES with the 2nd key and then encrypts the data a final time using DES and the 3rd key. The reason for the creation of 3DES is that DES alone was no-longer resistant to various attacks. DES works by expanding the 32-bit half block to 8 6-bit blocks (total of 48-bit). The 48-bit key block is then xored with a 48-bit subkey. The 8 sub-keys then go through a non-liner substitution box and then finally go through a permutation box. None of the features within DES stand up to current encryption standards and 3DES is just a repetition of Des therefor, none of the features within DES can be used to improve or further increase the security if implemented into this project's algorithm.

Rivest Cipher 4

Rivest Cipher 4 (RC4) is a stream cypher, this means that the algorithm works on an array of data rather than a 2D block, this makes the cypher easy to expand indefinitely. RC4 is incredibly mathematically complex and is difficult to understand how the algorithm functions. This algorithm has been retired by many users such as Microsoft and Google because RC4 is no longer secure (Peter Loshin, 2016). Although this algorithm is insecure and has many vulnerabilities there are numerous other algorithms that have used RC4 as a building block, these include Spritz, RC4A, VMPC and RC4+. Spritz is a stream cypher proposed by Rivest the creator of RC4 and Schuldt as a replacement to RC4. As these cyphers are stream cyphers, they provide insight into an alternative type of encryption however their features are incompatible with a block type cypher such as AES. (*GeekforGeeks, 2020*)

Design

Tools

To make this project a reality a few tools will be needed. Firstly, an Integrated Development Environment (IDE) that supports python will be required, the chosen software is Integrated Development and Learning Environment 3 (IDLE3). A text editor is also required so that a report can be written, Microsoft Word was chosen as it is a powerful word processor an because a product licence has already been acquired. Google has been selected as the search engine to research and to find python commands. Paint.net is also required to create diagrams this is due to previous experience with the package.

Programming Language

To code the project, the decision was made to use python as it is one of the faster higher-level programming languages and is notorious for being one of the fastest languages to code in. this is because python is a dynamically typed language. This means python does not know the variable types until runtime. This can increase the risk of errors but can be mitigated by using functions to convert data types e.g. str() or int(). Python also has simpler syntax when compared with other programming languages such as C, C#, C++ and Java. (*Guru99, No date*)

Increased Key Sizes

The standard for AES only includes 3 key sizes 128-bit, 192 bit and 256 bit. By increasing this to 384 bit and 512 bit. Increasing the key size makes the encryption harder to brute force as more guesses are needed. For AES 256-bit 2^{256} possible keys or 1.1579209e+77 for 1024 bit there are 2^{1024} possible combinations or 1.79769313e+308.

Interlaced Keys

When looking at other cyphers on a mobile phone a comment in an internet forum was found. The forum went into detail about AES being able to get to 512-bit. The reason for this is that they had used two initial keys and ran key expansion on both to get a bigger block of round keys. It gave an idea to interlace 2 of the increased sized keys from

before. This means each round of encryption will alternate the use of keys from two differently generated keys that are independent of each other.

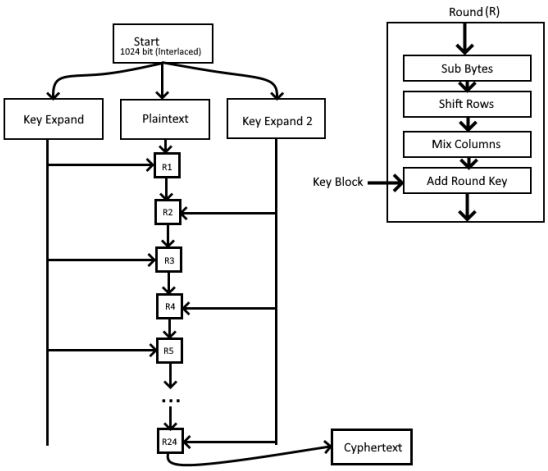


Diagram showing the two independent keys being used for alternating rounds.

Using this new method should add diffusion into the algorithm as now there's not just 1 key to reverse but 2, if the attacker doesn't know this method is being used, they won't be able to retrieve any key as if they get a key it will be an invalid mix of key A and B.

User Interface

The program will have a text-based User Interface (UI) this will include input options for the key size, the key data, and the plaintext data. This program will only calculate one block of plaintext and will show the encrypted block and then the encrypted block decrypted. This will then allow the program to output data and show the new algorithm works and allow debugging, this will also save time creating a GUI and adding user input handling. (*Computer Hope, 2021*)

Prototype

Main Functions

Block Class

```
class block:
    def __init__(self,size = 128): #set up 4x4 array
        self.data = [["","","",""],["","","",""],["","","",""],["","","",""]]
        self.size = size
    def display(self): # print the 4x4 display to screen (debugging)
        print("-------------")
        for i in range(0,4):
            print("|" + self.data[i][0] + "|" + self.data[i][1] + "|" + self.data[i][2] + "|" + self.data[i][3] + "|")
            print("-------------")
    def getbincell(self,x,y): #get the binary value of a single cell
        val = self.data[x][y]
        ret = ""
        for i in range(0,16):
            if val[0] == convhex[i][0]:
                ret+=convhex[i][1]
        for i in range(0,16):
            if val[1] == convhex[i][0]:
                ret+=convhex[i][1]
        return ret
    def getraw(self): # get the binary value of the entier block
        val = ""
        ret = ""
        for x in range(0,4):
            for y in range(0,4):
                val = self.data[x][y]
                for b in range(0,2):
                    for i in range(0,16):
                        if val[b] == convhex[i][0]:
                            ret+=convhex[i][1]
        return ret
    def hexset(self,hexin):
        count=0
        for x in range(0,4):
            for y in range(0,4):
                self.data[y][x] = hexin[count] + hexin[count+1]
                count+=2

class bigblock:
    def __init__(self,size = 128): #set up 4x4 array
        self.data = [["","","","","","","","","","","","","","","",""],["","","","","","","","","","","","","","","",""],
```

The Block and Bigblock classes

The block class is used to store the data for encryption as well as the keys. The block function includes a 4x4 array that is designed to hold two hexadecimal characters. The block class also includes some useful functions to get and set data in different formats, as well as the display function which, which shows the current contents of the cell in an easy-to-read way. The bigblock class is used for key expansion for any key size above 128-bit, which can store up to 16 columns of data.

SubBytes

```
sbox = [["63","7C","77","7B","F2","6B","6F","C5","30","01","67","2B","FE","D7","AB","76"],["CA","82","C9","7D","FA","59
invsbox = [["52","09","6A","D5","30","36","A5","38","BF","40","A3","9E","81","F3","D7","FB"],["7C","E3","39","82","9B",
```

The sbox and inverse sbox declared at the beginning of the program (could not fit on the screen)

The sbox contains a 16x16 array of predetermined values as declared in the AES 2001 standard.

```
def SubBytes(block_data,crypttype):#Substitute Cells For SBOX
    for x in range (0,4):
        for y in range (0,4):
            current = block_data.data[x][y] #Get Each Cell
            cord = getCord(current) # Get Its Coordinates In SBOX
            if (crypttype == "E"): # If Encrypting
                block_data.data[x][y] = sbox[cord[0]][cord[1]] # Return SBOX Value To Cell
            elif (crypttype == "D"):#If Decrypting
                block_data.data[x][y] = invsbox[cord[0]][cord[1]]  # Return SBOX Value To Cell
    return 0 #Return Success
```

The SubBytes Function

The SubBytes function was easy to code as the function it performs is very basic however a separate function was made to retrieve the coordinates this is called getCord. The input variables are block data which contains the plaintext data in a 4x4 array and crypttype which is used to tell if the program is encrypting or decrypting the data and is

represented by a simple “E” or “D”. As the function changes the block data directly the only return data is 0 this can be used to help bugfix as it tells program the function has successfully exited.

```
def getCord(text):#Turns A 2 Diget Hex Value Into Decimal Coordinates
    cord = [0,0]
    for i in range(0,2):
        num = text[i]
        if (num == "A"):
            cord[i] = 10
        elif (num == "B"):
            cord[i] = 11
        elif (num == "C"):
            cord[i] = 12
        elif (num == "D"):
            cord[i] = 13
        elif (num == "E"):
            cord[i] = 14
        elif (num == "F"):
            cord[i] = 15
        else:
            cord[i] = int(num)
    #print(str(cord[0])+" "+str(cord[1])) #DEBUG REMOVE
    return cord
```

The getCord function

The getCord function is not included in the official 2001 AES documentation. this function was created to solve the problem of converting hexadecimal to base 10 and formatting it in a way that is easy to manipulate. The code takes 2 hexadecimal characters and if their value is over 9 turns them into base 10 and adds them to an array with 2 cells 1 for each hexadecimal character e.g., C3 = [12,3]. These coordinates are then returned to the calling function.

ShiftRows

```
def ShiftRows(block_data,crypttype): #Shift Rows Across
    temp_block=[["00","00","00","00"],["00","00","00","00"],["00","00","00","00"],["00","00","00","00"]]#temp empty block
    temp_block[0] = block_data.data[0] # set the first row of the empty block to the first row of the current block
    for y in range(0,4):
        for x in range(0,4): # for each cell
            if (crypttype == "E"): # if encrypting
                if (x+y<=3): # if within range 0-3
                    temp_block[y][x]=block_data.data[y][x+y]# shift cell by y e.g. further down the block more it moves across
                else:
                    temp_block[y][x]=block_data.data[y][(x+y)-4]# shift cell with correction for out of range
            elif (crypttype == "D"):# if decrypting
                if (x-y >=0):# if within range 0-3
                    temp_block[y][x]=block_data.data[y][x-y]# shift cell by y e.g. further down the block more it moves across
                else:
                    temp_block[y][x]=block_data.data[y][(x-y)+4]# shift cell with correction for out of range
    block_data.data=temp_block# set current block to the results
    return 0
```

The Shift Rows function

The ShiftRows function was also easy to code because as the y value changes the and the amount the cells move left is the same. This means that the cells can be set to their new position in a loop that run for each cell. As with SubBytes, the inputs for this function are the current encryption block and whether the function is encrypting or decrypting. This function also returns 0 to help with debugging.

MixColumns

```
def MixColumns(block_data,crypttype):
    blocknew = [["","","",""],["","","",""],["","","",""],["","","",""]] # empty block
    Cval = ["","","",""] # Colum values
    Gval = [0,0,0,0] # Given Value
    for y in range(0,4):
        for x in range(0,4):
            if (crypttype == "E"): # If encrypting
                Gval = mixgrid[y] # Get the row from mixgrid
            elif (crypttype == "D"): # If decrypting
                Gval = invmixgrid[y] # Get the row from inverted mixgrid
            for i in range (0,4): # For Each Cell within the column
                val = block_data.data[i][x] # Current cell
                index = getDecimal(val) # get the decimal value for the cell
                if (Gval[i] == 1): # for x1
                    Cval[i] = val # set cval value to the cell
                elif (Gval[i] == 2): # for x2
                    Cval[i] = multi2[index] # set cval value to one from the lookup table multi2
                elif (Gval[i] == 3): # for x3
                    Cval[i] = multi3[index] # set cval value to one from the lookup table multi3
                elif (Gval[i] == 9): # for x9
                    Cval[i] = multi9[index] # set cval value to one from the lookup table multi9
                elif (Gval[i] == 11): # for x11
                    Cval[i] = multi11[index] # set cval value to one from the lookup table multi11
                elif (Gval[i] == 13): # for x13
                    Cval[i] = multi13[index] # set cval value to one from the lookup table multi13
                elif (Gval[i] == 14): # for x14
                    Cval[i] = multi14[index] # set cval value to one from the lookup table multi14
            new = mod2Add4(Cval) #xor values and set as new
            blocknew[y][x] = new # set newblock cell to new
    block_data.data = blocknew # set current block to the blocknew
```

The MixColumns function

The mixcolums at first appeared difficult to code, however, after looking online a website suggested using lookup tables (Cryptography Fandom, no date) to avoid having to do the complex maths for Galois fields.

```
multi2 = ["00","02","04","06","08","0A","0C","0E","10","12","14","16","18","1A","1C","1E","20","22","24","26","28","2A","2C"
multi3 = ["00","03","06","05","0C","0F","0A","09","18","1B","1E","1D","14","17","12","11","30","33","36","35","3C","3F","3A"
multi9 = ["00","09","12","1B","24","2D","36","3F","48","41","5A","53","6C","65","7E","77","90","99","82","8B","B4","BD","A6"
multi11 = ["00","0B","16","1D","2C","27","3A","31","58","53","4E","45","74","7F","62","69","B0","BB","A6","AD","9C","97","8A
multi13 = ["00","0D","1A","17","34","39","2E","23","68","65","72","7F","5C","51","46","4B","D0","DD","CA","C7","E4","E9","FE
multi14 = ["00","0E","1C","12","38","36","24","2A","70","7E","6C","62","48","46","54","5A","E0","EE","FC","F2","D8","D6","C4
```

Lookup tables for Galois field multiplication (couldn't fit on the screen)

A separate function was created for performing xor operations as the key expansion function also performs them. As with the last 2 functions, the inputs for this function are the current encryption block and whether the function is encrypting or decrypting.

```
def mod2Add4(numbers):#  format ["FF","FF","FF","FF"]
    binaryval = ["","","",""] #binary format of above ["11111111","11111111","11111111","11111111"]
    for i in range(0,4):#for each byte
        no = ""
        for b in range(0,2):#for each 4bit
            for c in range(0,16):#for each letter in hex
                if numbers[i][b] == convhex[c][0]: #if same as lookup
                    binaryval[i]+=convhex[c][1]#set binary val
    current = binaryval[0] #set curretnt value to first value
    for i in range(1,4): # for each value starting with [1]
        add = binaryval[i]#current [i] value to xor
        for b in range(0,8): #for each binary value
            if (int(current[b]) + int(add[b]) <2): # if it adds to less than 2
                calc = str(int(current[b]) + int(add[b]))# add
                current = modstr(current,b,calc)# replace bit with added value
            else:
                current = modstr(current,b,"0")#replace bit with 0
        if (i==3):#on the last round
            no1 = "" #1st hex letter
            no2 = ""#2nd hex letter
            ret = ""#to return
            for c in range (0,8):#for each binary value
                if (c<4):#for each hex char
                    no1 += current[c]#add to no1
                else:
                    no2 += current[c]#add to no2
            for d in range (0,16):#for each letter in hex
                if no1 == convhex[d][1]:#if same as hex lookup
                    no1 = convhex[d][0] #set letter
                if no2 == convhex[d][1]:#if same as hex lookup
                    no2 = convhex[d][0] #set letter
            ret = no1 + no2 #set to return
            return ret #return xored value
```

Function For Xoring

A fuction was created for xoring up to 4 hexadecimal numbers. This took a long time to create as the function first has to convert the hexadecimal inputs to binary before xoring and then has to convert them back afterwards, it does this using a basic lookup table to avoid long amounts of code. There is also another function within mod2Add4 called modstr.

```
def modstr(string,pos,repl):#modify a single char in a string by index
    newstr = "" #new string
    for i in range(0,len(string)):#for each char
        if i == pos:#if at index
            newstr += repl#add replacement char
        else:
            newstr += string[i]#add normal char
    return newstr

```

The Modstr function

In python there is not a way to change a single character in a string based on its index, so one was created. The function is fairly simple just adding characters to a string until the index is reached where it adds the new character before continuing. It takes 3 inputs the string that needs changing, the index (pos) and the replacement (repl), The function then returns the new string.

```
convhex = [["0","0000"],["1","0001"],["2","0010"],["3","0011"],["4","0100"],["5","0101"],["6","0110"],["7","0111"],["8","1000"],["9","1001"],
convdec = [["0",0],["1",1],["2",2],["3",3],["4",4],["5",5],["6",6],["7",7],["8",8],["9",9],["A",10],["B",11],["C",12],["D",13],["E",14],["F",
```

The lookup tables for converting hex to binary and decimal (could not fit on the screen)

AddRoundKey

```
def AddRoundKey(block_data,block_key):
    newblock = [["","","",""],["","","",""],["","","",""],["","","",""]]# new empty block
    for x in range(0,4):
        for y in range (0,4): #for each cell
            newblock[x][y] = mod2Add4([block_data.data[x][y],block_key.data[x][y],"00","00"])# set newblock cell to exor of two cells from key and data
    block_data.data = newblock# set current block to newblock
```

The AddRoundKey function

The AddRoundKey function was very easy to create as a function for xoring was already created, so the function just sends each cell from the current encryption block and current round key and xors them together. This function only takes the encryption block and a key block as its inputs. Because of how xor works if a value is xored twice the value started with is returned e.g., 101 xor 010 = 111 and in reverse 111 xor 010 = 101. For this reason, there is no need to tell the function whether it is encrypting or decrypting.

KeyExpand

128 Bit

```
def KeyExpand(keydata, Size,hexmode):
    key0 = SettupKey0(keydata,Size,hexmode) #sets up first key block off user input
    if (Size == 128): #if 128 bit encryption
        key.append(key0) #append the first key to key list
    else: #if above 128 bit encryption
        bigkey.append(key0)#append the first key to key list
    #128 Bit
    if (Size == 128):
        for i in range(0,10):#for each round of encryption
            lastkey = key[i]#the previous key block
            newkey = block()# make a new key block
            w0 = [lastkey.data[0][0],lastkey.data[1][0],lastkey.data[2][0],lastkey.data[3][0]]#column 0
            w1 = [lastkey.data[0][1],lastkey.data[1][1],lastkey.data[2][1],lastkey.data[3][1]]#column 1
            w2 = [lastkey.data[0][2],lastkey.data[1][2],lastkey.data[2][2],lastkey.data[3][2]]#column 2
            w3 = [lastkey.data[0][3],lastkey.data[1][3],lastkey.data[2][3],lastkey.data[3][3]]#column 3
            gval = g(w3,i)#run function g on the last column
            w4 = XorCollum(w0,gval)#column 0 in the new block = xor column 0 and gval
            w5 = XorCollum(w4,w1)#column 1 in the new block = xor column 0(new) and gval
            w6 = XorCollum(w5,w2)#column 2 in the new block = xor column 1(new) and gval
            w7 = XorCollum(w6,w3)#column 3 in the new block = xor column 2(new) and gval
            new = [w4,w5,w6,w7]# add new columns to make a new block
            for x in range(0,4):
                for y in range(0,4):# for each cell
                    newkey.data[x][y] = new[y][x]# set the cell to the new value
```

The KeyExpand function (128-bit)

This function was easy to code however with the requirement of other key sizes it ended up getting very lengthy. 3 functions were created for key expansion SettupKey0, XorCollum and g. KeyExpand takes 3 inputs the key the user inputs called keydata, the key size just named size and hexmode which decides whether the user input is hexadecimal or a string.

```
242 def XorRcon(c,r): #xors the first cell with the round rcon value
243     c[0] = mod2Add4([rcon[r],c[0],"00","00"])
244     return c
245 
246 def RotWord(c): #rearanges the cells
247     return [c[1],c[2],c[3],c[0]]
248 
249 def SubWord(c):
250     for x in range (0,4):
251         current = c[x] #Get Each Cell
252         cord = getCord(current) # Get Its Coordinates In SBOX
253         c[x] = sbox[cord[0]][cord[1]] # Return SBOX Value To Cell
254     return c
255 
256 def g(c,r): #runs rotword subword and xorrcon on a column
257     v0 = RotWord(c)
258     v1 = SubWord(v0)
259     v2 = XorRcon(v1,r)
260     return v2
```

The g function and its sub-functions XorRcon, Rotword and SubWord

The g function on its own just runs a column through several functions. The inputs for g are a column of 4 cells and an integer with the current round.

Rotworld is a simple function that just re-arranges the cells from 1234 to 2341.

SubWord runs a simple version of the SubBytes function using the value of a cell as coordinates for a new value in a substitution box this function, however, is much shorter due to the fact it does not need to be able to undo the substitution and only runs on 4 cells.

```
262 def SettupKey0(key,size,hexmode):#format first key
263     if (size == 128):#if 128-bit
264         newblock = [["2B", "28", "AB", "09"],["7E", "AE", "F7", "CF"],["15", "D2", "15", "4F"],["16
265         if hexmode:#if input is in hex
266             count=0#counter
267             for x in range(0,4):
268                 for y in range(0,4):#for each cell
269                     newblock[y][x] = key[count] + key[count+1]#add 2 hex chars to cell
270                     count+=2#increment count
271         else:#if input is string
272             c = -1#counter
273             for y in range(0,4):
274                 for x in range(0,4):#for each cell
275                     c += 1#increment count
276                     letter = key[c]#get the current letter in the string
277                     newblock[x][y] = CharToHex(letter)#convert letter to hexadecimal and set cell
278         key0 = block()#create new key
279         key0.data = newblock#set its data
280         return key0#return the new key block
```

The SettupKey0 Function

This function sets up the first key from the key the user input. This function takes 3 inputs the key the user input named key, the key size called size and the input mode called hexmode. The function takes the user inputted key and turns it into hexadecimal format if it is not already. Then it puts the key into a block the same size as the key size.

Key Size (bits)	Block size
128	4x4

192	6x4
256	8x4
384	12x4
512	16x4
1024 (interlaced)	2x16x4

all blocks are turned into 128-bit blocks for the encryption process after sub-keys are made.

```
355 def XorCollum(c1,c2):
356     r0 = mod2Add4([c1[0],c2[0],"00","00"])
357     r1 = mod2Add4([c1[1],c2[1],"00","00"])
358     r2 = mod2Add4([c1[2],c2[2],"00","00"])
359     r3 = mod2Add4([c1[3],c2[3],"00","00"])
360     return [r0,r1,r2,r3]
```

The Xorcollum function

This function just helps keep the code for xoring on one line. The function xors two columns cell by cell then returns the finished column.

192 bit

```
390     # 192
391     elif (Size == 192):
392         for i in range(0,8):#for each round of encryption
393             lastkey = bigkey[i]#the previous key block
394             newkey = bigblock()# make a new key block
395             w0 = [lastkey.data[0][0],lastkey.data[1][0],lastkey.data[2][0],lastkey.data[3][0]]#column 0
396             w1 = [lastkey.data[0][1],lastkey.data[1][1],lastkey.data[2][1],lastkey.data[3][1]]#column 1
397             w2 = [lastkey.data[0][2],lastkey.data[1][2],lastkey.data[2][2],lastkey.data[3][2]]#column 2
398             w3 = [lastkey.data[0][3],lastkey.data[1][3],lastkey.data[2][3],lastkey.data[3][3]]#column 3
399             w4 = [lastkey.data[0][4],lastkey.data[1][4],lastkey.data[2][4],lastkey.data[3][4]]#column 4
400             w5 = [lastkey.data[0][5],lastkey.data[1][5],lastkey.data[2][5],lastkey.data[3][5]]#column 5
401             gval = g(w5,i)#run function g on the last column
402             w6 = XorCollum(w0,gval)#column 0 in the new block = xor column 0 and gval
403             w7 = XorCollum(w6,w1)#column 1 in the new block = xor column 0(new) and gval
404             w8 = XorCollum(w7,w2)#column 2 in the new block = xor column 1(new) and gval
405             w9 = XorCollum(w8,w3)#column 3 in the new block = xor column 2(new) and gval
406             w10 = XorCollum(w9,w4)#column 4 in the new block = xor column 3(new) and gval
407             w11 = XorCollum(w10,w5)#column 5 in the new block = xor column 4(new) and gval
408             new = [w6,w7,w8,w9,w10,w11]#add new columns to make a new block
409             for x in range(0,4):
410                 for y in range(0,6):#for each cell
411                     newkey.data[x][y] = new[y][x]# set the cell to the new value
412             newkey.size = 192 #set size
413             bigkey.append(newkey)#append the key
```

192-bit key expansion

For 192-bit key expansion, there are six columns. This creates a problem as 6 is not directly divisible by 4. To solve this issue another chunk of code corrects this using two blocks for 12 columns which are devisable by 4.

```
    if (Size == 192): #Fixing 6 Column Block To 4 Column
        for i in range(0,len(bigkey)):#for all 192 bit keys
            currentkey = bigkey[i] # the current selected key
            newkey  = block()#new regular block
            newkey2  = block()#new regular block
            newkey.size = 192#set 192 bit
            newkey2.size = 192#set 192 bit
            if(i%2 == 1):# if i is an odd number
                lastkey = bigkey[i-1]#the previous key
                w0 = [lastkey.data[0][4],lastkey.data[1][4],lastkey.data[2][4],lastkey.data[3][4]]#get last 2 columns from the privious key
                w1 = [lastkey.data[0][5],lastkey.data[1][5],lastkey.data[2][5],lastkey.data[3][5]]
                w2 = [currentkey.data[0][0],currentkey.data[1][0],currentkey.data[2][0],currentkey.data[3][0]]# 6 colums from this key
                w3 = [currentkey.data[0][1],currentkey.data[1][1],currentkey.data[2][1],currentkey.data[3][1]]
                w4 = [currentkey.data[0][2],currentkey.data[1][2],currentkey.data[2][2],currentkey.data[3][2]]
                w5 = [currentkey.data[0][3],currentkey.data[1][3],currentkey.data[2][3],currentkey.data[3][3]]
                w6 = [currentkey.data[0][4],currentkey.data[1][4],currentkey.data[2][4],currentkey.data[3][4]]
                w7 = [currentkey.data[0][5],currentkey.data[1][5],currentkey.data[2][5],currentkey.data[3][5]]
                new = [w0,w1,w2,w3]#set regular block
                new2 = [w4,w5,w6,w7]#set regular block
                for x in range(0,4):
                    for y in range(0,4):#for each cell
                        newkey.data[x][y] = new[y][x]#set data from new
                        newkey2.data[x][y]=new2[y][x]#set data from new

                key.append(newkey)#append normal key
                key.append(newkey2)#append normal key
            else:#if i is even
                w0 = [currentkey.data[0][0],currentkey.data[1][0],currentkey.data[2][0],currentkey.data[3][0]]#get the first 4 columns from block
                w1 = [currentkey.data[0][1],currentkey.data[1][1],currentkey.data[2][1],currentkey.data[3][1]]
                w2 = [currentkey.data[0][2],currentkey.data[1][2],currentkey.data[2][2],currentkey.data[3][2]]
                w3 = [currentkey.data[0][3],currentkey.data[1][3],currentkey.data[2][3],currentkey.data[3][3]]
                newkey.size = 128
                new = [w0,w1,w2,w3]#set new as data
                for x in range(0,4):
                    for y in range(0,4):#for each cell
                        newkey.data[x][y] = new[y][x]# set data
                key.append(newkey)#append key
```

6 column fix

This goes through each 6-column key and if the index for the key in the key array is odd it grabs the last two columns from the previous key, and if the index is even, it only uses the first 4 columns, Then it appends the columns as 182-bit keys.

Commented [JC1]: Check Paragraph

256 bit

```
    elif (Size == 256):
        newkey = block()# make a new key block
        newkey2 = block()# make a new key block
        key0 = bigkey[0]#key 0
        w0 = [key0.data[0][0],key0.data[1][0],key0.data[2][0],key0.data[3][0]]#get all columns
        w1 = [key0.data[0][1],key0.data[1][1],key0.data[2][1],key0.data[3][1]]
        w2 = [key0.data[0][2],key0.data[1][2],key0.data[2][2],key0.data[3][2]]
        w3 = [key0.data[0][3],key0.data[1][3],key0.data[2][3],key0.data[3][3]]
        w4 = [key0.data[0][4],key0.data[1][4],key0.data[2][4],key0.data[3][4]]
        w5 = [key0.data[0][5],key0.data[1][5],key0.data[2][5],key0.data[3][5]]
        w6 = [key0.data[0][6],key0.data[1][6],key0.data[2][6],key0.data[3][6]]
        w7 = [key0.data[0][7],key0.data[1][7],key0.data[2][7],key0.data[3][7]]
        new = [w0,w1,w2,w3]#seperate into 2 regular blocks
        new2 = [w4,w5,w6,w7]
        for x in range(0,4):
            for y in range(0,4):
                newkey.data[x][y] = new[y][x]
                newkey2.data[x][y] = new2[y][x]#set the values into the new keys

        key.append(newkey)#append the keys
        key.append(newkey2)
        for i in range(0,7):#for each round of encryption
            lastkey = bigkey[i]#the previous key block
            newkey = block()#new block
            newkey2 = block()#new block
            newkey3 = bigblock()#new big block
            w0 = [lastkey.data[0][0],lastkey.data[1][0],lastkey.data[2][0],lastkey.data[3][0]]#get all columns
            w1 = [lastkey.data[0][1],lastkey.data[1][1],lastkey.data[2][1],lastkey.data[3][1]]
            w2 = [lastkey.data[0][2],lastkey.data[1][2],lastkey.data[2][2],lastkey.data[3][2]]
            w3 = [lastkey.data[0][3],lastkey.data[1][3],lastkey.data[2][3],lastkey.data[3][3]]
            w4 = [lastkey.data[0][4],lastkey.data[1][4],lastkey.data[2][4],lastkey.data[3][4]]
            w5 = [lastkey.data[0][5],lastkey.data[1][5],lastkey.data[2][5],lastkey.data[3][5]]
            w6 = [lastkey.data[0][6],lastkey.data[1][6],lastkey.data[2][6],lastkey.data[3][6]]
            w7 = [lastkey.data[0][7],lastkey.data[1][7],lastkey.data[2][7],lastkey.data[3][7]]
            gval = g(w7,i)#run function g on the last column
            w8 = XorCollum(w0,gval)# xor first column of previous key with gval
            w9 = XorCollum(w8,w1) #xor columns from previous key with last column
            w10 = XorCollum(w9,w2)
            w11 = XorCollum(w10,w3)
            w11 = SubWord(w11) #Subword on 4th column
            w12 = XorCollum(w11,w4)
            w13 = XorCollum(w12,w5)
            w14 = XorCollum(w13,w6)
            w15 = XorCollum(w14,w7)
            w11 = XorCollum(w10,w3)#reset column
            new = [w8,w9,w10,w11]#set values in block
            new2 = [w12,w13,w14,w15]#set values in block
            new3 = [w8,w9,w10,w11,w12,w13,w14,w15] #set one big block
            for x in range(0,4):
                for y in range(0,8):
                    newkey3.data[x][y] = new3[y][x]#set cells
            for x in range(0,4):
                for y in range(0,4):
                    newkey.data[x][y] = new[y][x]#set cells
                    newkey2.data[x][y] = new2[y][x]#set cells
            newkey.size = 128 #set sizes
            newkey2.size = 128
            newkey3.size = 256
            key.append(newkey)#append keys
            key.append(newkey2)
            bigkey.append(newkey3)
```

256-bit keys are split into 2 128-bit keys. The 256-bit key expansion introduces the subword function (used in the g function) on the 4th column, when testing 256-bit encryption was tried against the test vectors in the 2001 AES standard the answer did not match. Setting the column back to its original value after the g function was ran was tried and it matched the test variables. This means the subword used on column 4 is only used to calculate the other columns and do not remain the 4th column in the final key.

384 bit

```
    elif (Size == 384):
        newkey = block()#make new blocks
        newkey2 = block()
        newkey3 = block()
        key0 = bigkey[0]
        w0 = [key0.data[0][0],key0.data[1][0],key0.data[2][0],key0.data[3][0]]#get all columns
        w1 = [key0.data[0][1],key0.data[1][1],key0.data[2][1],key0.data[3][1]]
        w2 = [key0.data[0][2],key0.data[1][2],key0.data[2][2],key0.data[3][2]]
        w3 = [key0.data[0][3],key0.data[1][3],key0.data[2][3],key0.data[3][3]]
        w4 = [key0.data[0][4],key0.data[1][4],key0.data[2][4],key0.data[3][4]]
        w5 = [key0.data[0][5],key0.data[1][5],key0.data[2][5],key0.data[3][5]]
        w6 = [key0.data[0][6],key0.data[1][6],key0.data[2][6],key0.data[3][6]]
        w7 = [key0.data[0][7],key0.data[1][7],key0.data[2][7],key0.data[3][7]]
        w8 = [key0.data[0][8],key0.data[1][8],key0.data[2][8],key0.data[3][8]]
        w9 = [key0.data[0][9],key0.data[1][9],key0.data[2][9],key0.data[3][9]]
        w10 = [key0.data[0][10],key0.data[1][10],key0.data[2][10],key0.data[3][10]]
        w11 = [key0.data[0][11],key0.data[1][11],key0.data[2][11],key0.data[3][11]]
        new = [w0,w1,w2,w3]#set data
        new2 = [w4,w5,w6,w7]
        new3 = [w8,w9,w10,w11]
        for x in range(0,4):
            for y in range(0,4):
                newkey.data[x][y] = new[y][x]#set cells
                newkey2.data[x][y] = new2[y][x]
                newkey3.data[x][y] = new3[y][x]

        key.append(newkey)#append keys
        key.append(newkey2)
        key.append(newkey3)
        for i in range(0,6):#for each round (previously key0)
            lastkey = bigkey[i]#last key
            newkey = block()#new key blocks
            newkey2 = block()
            newkey3 = block()
            newkey4 = bigblock()
            w0 = [lastkey.data[0][0],lastkey.data[1][0],lastkey.data[2][0],lastkey.data[3][0]]#get columns
            w1 = [lastkey.data[0][1],lastkey.data[1][1],lastkey.data[2][1],lastkey.data[3][1]]
            w2 = [lastkey.data[0][2],lastkey.data[1][2],lastkey.data[2][2],lastkey.data[3][2]]
            w3 = [lastkey.data[0][3],lastkey.data[1][3],lastkey.data[2][3],lastkey.data[3][3]]
            w4 = [lastkey.data[0][4],lastkey.data[1][4],lastkey.data[2][4],lastkey.data[3][4]]
            w5 = [lastkey.data[0][5],lastkey.data[1][5],lastkey.data[2][5],lastkey.data[3][5]]
            w6 = [lastkey.data[0][6],lastkey.data[1][6],lastkey.data[2][6],lastkey.data[3][6]]
            w7 = [lastkey.data[0][7],lastkey.data[1][7],lastkey.data[2][7],lastkey.data[3][7]]
            w8 = [lastkey.data[0][8],lastkey.data[1][8],lastkey.data[2][8],lastkey.data[3][8]]
            w9 = [lastkey.data[0][9],lastkey.data[1][9],lastkey.data[2][9],lastkey.data[3][9]]
            w10 = [lastkey.data[0][10],lastkey.data[1][10],lastkey.data[2][10],lastkey.data[3][10]]
            w11 = [lastkey.data[0][11],lastkey.data[1][11],lastkey.data[2][11],lastkey.data[3][11]]
            gval = g(w11,i)#run function g on the last column
            w12 = XorCollum(w0,gval)# xor first column of previous key with gval
            w13 = XorCollum(w12,w1)#xor columns from previous key with last column
            w14 = XorCollum(w13,w2)
            w15 = XorCollum(w14,w3)
            w15 = SubWord(w15)#Subword on 4th column
            w16 = XorCollum(w15,w4)
            w17 = XorCollum(w16,w5)
            w18 = XorCollum(w17,w6)
            w19 = XorCollum(w18,w7)
            w19 = SubWord(w19)#Subword on 8th column
            w20 = XorCollum(w19,w8)
            w21 = XorCollum(w20,w9)
            w22 = XorCollum(w21,w10)
```

```
            w23 = XorCollum(w22,w11)
            w15 = XorCollum(w14,w3)#reset column
            w19 = XorCollum(w18,w7)#reset column
            new = [w12,w13,w14,w15]#set data
            new2 = [w16,w17,w18,w19]
            new3 = [w20,w21,w22,w23]
            new4 = [w12,w13,w14,w15,w16,w17,w18,w19,w20,w21,w22,w23]
            for x in range(0,4):
                for y in range(0,12):
                    newkey4.data[x][y] = new4[y][x]#set cells
            for x in range(0,4):
                for y in range(0,4):
                    newkey.data[x][y] = new[y][x]#set cells
                    newkey2.data[x][y] = new2[y][x]
                    newkey3.data[x][y] = new3[y][x]
            newkey.size = 128#set size
            newkey2.size = 128
            newkey3.size = 128
            newkey4.size = 384
            key.append(newkey)#append keys
            key.append(newkey2)
            key.append(newkey3)
            bigkey.append(newkey4)
```

384 bit is not included in the official documentation and is the first of the changes to improve AES. 384 is the same as 3 128-bit keys merged taking up 12 columns and uses two subword functions outside the g function, on the 4th and 8th column. After working out the next key block it is cut into 3 128-bit keys.

512 bit

```
    elif (Size == 512):
        newkey = block()#new key blocks
        newkey2 = block()
        newkey3 = block()
        newkey5 = block()
        key0 = bigkey[0]
        w0 = [key0.data[0][0],key0.data[1][0],key0.data[2][0],key0.data[3][0]]#get columns
        w1 = [key0.data[0][1],key0.data[1][1],key0.data[2][1],key0.data[3][1]]
        w2 = [key0.data[0][2],key0.data[1][2],key0.data[2][2],key0.data[3][2]]
        w3 = [key0.data[0][3],key0.data[1][3],key0.data[2][3],key0.data[3][3]]
        w4 = [key0.data[0][4],key0.data[1][4],key0.data[2][4],key0.data[3][4]]
        w5 = [key0.data[0][5],key0.data[1][5],key0.data[2][5],key0.data[3][5]]
        w6 = [key0.data[0][6],key0.data[1][6],key0.data[2][6],key0.data[3][6]]
        w7 = [key0.data[0][7],key0.data[1][7],key0.data[2][7],key0.data[3][7]]
        w8 = [key0.data[0][8],key0.data[1][8],key0.data[2][8],key0.data[3][8]]
        w9 = [key0.data[0][9],key0.data[1][9],key0.data[2][9],key0.data[3][9]]
        w10 = [key0.data[0][10],key0.data[1][10],key0.data[2][10],key0.data[3][10]]
        w11 = [key0.data[0][11],key0.data[1][11],key0.data[2][11],key0.data[3][11]]
        w12 = [key0.data[0][12],key0.data[1][12],key0.data[2][12],key0.data[3][12]]
        w13 = [key0.data[0][13],key0.data[1][13],key0.data[2][13],key0.data[3][13]]
        w14 = [key0.data[0][14],key0.data[1][14],key0.data[2][14],key0.data[3][14]]
        w15 = [key0.data[0][15],key0.data[1][15],key0.data[2][15],key0.data[3][15]]
        new = [w0,w1,w2,w3]#set data
        new2 = [w4,w5,w6,w7]
        new3 = [w8,w9,w10,w11]
        new5 = [w12,w3,w14,w15]
        for x in range(0,4):
            for y in range(0,4):
                newkey.data[x][y] = new[y][x]#set cells
                newkey2.data[x][y] = new2[y][x]
                newkey3.data[x][y] = new3[y][x]
                newkey5.data[x][y] = new5[y][x]
        key.append(newkey)#append keys
        key.append(newkey2)
        key.append(newkey3)
        key.append(newkey5)
        for i in range(0,6):#for each round(previously key0)
            lastkey = bigkey[i]#last key
            newkey = block()#new key blocks
            newkey2 = block()
            newkey3 = block()
            newkey5 = block()
            newkey4 = bigblock()
            w0 = [lastkey.data[0][0],lastkey.data[1][0],lastkey.data[2][0],lastkey.data[3][0]]#get columns
            w1 = [lastkey.data[0][1],lastkey.data[1][1],lastkey.data[2][1],lastkey.data[3][1]]
            w2 = [lastkey.data[0][2],lastkey.data[1][2],lastkey.data[2][2],lastkey.data[3][2]]
            w3 = [lastkey.data[0][3],lastkey.data[1][3],lastkey.data[2][3],lastkey.data[3][3]]
            w4 = [lastkey.data[0][4],lastkey.data[1][4],lastkey.data[2][4],lastkey.data[3][4]]
            w5 = [lastkey.data[0][5],lastkey.data[1][5],lastkey.data[2][5],lastkey.data[3][5]]
            w6 = [lastkey.data[0][6],lastkey.data[1][6],lastkey.data[2][6],lastkey.data[3][6]]
            w7 = [lastkey.data[0][7],lastkey.data[1][7],lastkey.data[2][7],lastkey.data[3][7]]
            w8 = [lastkey.data[0][8],lastkey.data[1][8],lastkey.data[2][8],lastkey.data[3][8]]
            w9 = [lastkey.data[0][9],lastkey.data[1][9],lastkey.data[2][9],lastkey.data[3][9]]
            w10 = [lastkey.data[0][10],lastkey.data[1][10],lastkey.data[2][10],lastkey.data[3][10]]
            w11 = [lastkey.data[0][11],lastkey.data[1][11],lastkey.data[2][11],lastkey.data[3][11]]
            w12 = [key0.data[0][12],key0.data[1][12],key0.data[2][12],key0.data[3][12]]
            w13 = [key0.data[0][13],key0.data[1][13],key0.data[2][13],key0.data[3][13]]
            w14 = [key0.data[0][14],key0.data[1][14],key0.data[2][14],key0.data[3][14]]
            w15 = [key0.data[0][15],key0.data[1][15],key0.data[2][15],key0.data[3][15]]
            gval = g(w15,i)#run function g on the last column
            w16 = XorCollum(w0,gval)# xor first column of previous key with gval


            w17 = XorCollum(w16,w1)#xor columns from previous key with last column
            w18 = XorCollum(w17,w2)
            w19 = XorCollum(w18,w3)
            w19 = SubWord(w19)#Subword on 4th column
            w20 = XorCollum(w19,w4)
            w21 = XorCollum(w20,w5)
            w22 = XorCollum(w21,w6)
            w23 = XorCollum(w22,w7)
            w23 = SubWord(w23)#Subword on 8th column
            w24 = XorCollum(w23,w8)
            w25 = XorCollum(w24,w9)
            w26 = XorCollum(w25,w10)
            w27 = XorCollum(w26,w11)
            w27 = SubWord(w27)#Subword on 12th column
            w28 = XorCollum(w27,w8)
            w29 = XorCollum(w28,w9)
            w30 = XorCollum(w29,w10)
            w31 = XorCollum(w30,w11)
            #------------------
            w27 = XorCollum(w26,w11)#reset column
            w23 = XorCollum(w14,w3)#reset column
            w19 = XorCollum(w18,w7)#reset column
            new = [w16,w17,w18,w19]#set data
            new2 = [w20,w21,w22,w23]
            new3 = [w24,w25,w26,w27]
            new5 = [w28,w29,w30,w31]
            new4 = [w16,w17,w18,w19,w20,w21,w22,w23,w24,w25,w26,w27,w28,w29,w30,w31]
            for x in range(0,4):
                for y in range(0,12):
                    newkey4.data[x][y] = new4[y][x]#set cells
            for x in range(0,4):
                for y in range(0,4):
                    newkey.data[x][y] = new[y][x]#set cells
                    newkey2.data[x][y] = new2[y][x]
                    newkey3.data[x][y] = new3[y][x]
                    newkey5.data[x][y] = new5[y][x]
            newkey.size = 128#set size
            newkey2.size = 128
            newkey3.size = 128
            newkey4.size = 384
            newkey5.size = 128
            key.append(newkey)#append keys
            key.append(newkey2)
            key.append(newkey3)
            key.append(newkey5)
            bigkey.append(newkey4)
```

512 bit is not included in the official documentation and is the second of the changes to improve AES. 512 bit is the same as 4 128 bit keys merged or 16 columns and uses three subword functions outside the g function, on the 4th, 8th and 12th column. After working out the next key block it is cut into 4 128-bit keys. 512 bit is also used for the 1024 bit interlaced encryption.

Key Interlace Function

```
def KeyInterlace(keydata,keydata2,Size,hexmode):
    KeyExpand(keydata,Size,True)
    tempkey1=key.copy()
    key.clear()
    KeyExpand(keydata2,Size,True)
    tempkey2=key.copy()
    key.clear()
    for i in range(0,28):
        key.append(tempkey1[i])
        key.append(tempkey2[i])
```

The KeyInterlace function

This is the function that makes 1024-bit interlaced keys. It runs 2 512-bit key expansions for each one copying the subkey array then clearing the array to make way for the next one. After getting both subkey arrays into temporary arrays it then adds them to the key array with each array adding 1 or each roundkey to the new keyblock.

Start Function

```
def start():
    rounds = 10
    bit = 128
    print("Select Option:\n\r1.128Bit\n\r2.192Bit\n\r3.256Bit\n\r4.384Bit\n\r5.512Bit\n\r6.1024Bit(Interlaced)")
    answer = input()# SIZE SETUP
    if (answer == "1"):
        rounds = 10
        bit = 128
    elif (answer == "2"):
        rounds = 12
        bit = 192
    elif (answer == "3"):
        rounds = 14
        bit = 256
    elif (answer == "4"):
        rounds = 16
        bit = 384
    elif (answer == "5"):
        rounds = 18
        bit = 512
    elif (answer == "6"):
        rounds = 24
        bit = 512
    else:
        print("Invalid Answer")
        start()
        return(0)
        ################################################ KEY SETUP
    print("Hex Plaintext: ",end = "")
    plain = input()
    plain = plain.upper()
    print("Hex Key: ",end="")
    plainkey = input()
    plainkey = plainkey.upper()
    if(answer == "6"):
        print("Hex Key 2: ",end="")
        plainkey2 = input()
        plainkey2 = plainkey.upper()
    print("Raw Data: ")
    test = block()
    test.hexset(plain)
    test.display()
    print("Key: " + plainkey)
    if(answer == "6"):
        print("Key 2: " + plainkey2)
        print("2x",end=" ")
    print(str(bit)+" Bit")
    ################################################
    if (int(answer) <= 5):
        KeyExpand(plainkey,bit,True)
    elif(int(answer) > 5):
        KeyInterlace(plainkey,plainkey2,bit,True)
    ################################################
    AddRoundKey(test,key[0])
    for i in range(1,rounds):
        SubBytes(test,"E")
        ShiftRows(test,"E")
        MixColumns(test,"E")
        AddRoundKey(test,key[i])
    SubBytes(test,"E")
```

```
    ShiftRows(test,"E")
    AddRoundKey(test,key[rounds])
    test.display()
    print("Decrypt")
    AddRoundKey(test,key[rounds])
    ShiftRows(test,"D")
    SubBytes(test,"D")
    for i in range(rounds-1,0,-1):
        AddRoundKey(test,key[i])
        MixColumns(test,"D")
        ShiftRows(test,"D")
        SubBytes(test,"D")
    AddRoundKey(test,key[0])
    test.display()


#-----------------------------------------START-AREA-----------------------------------------------------------
start()
```

The start function.

The start function is the main function in the code where all other functions are called from. The start function starts by asking the user what key size they would like. After they have input a size it asks the user for their 182-bit plaintext along with their key and if interlaced keys their 2nd key as well. Once it has this information It starts by running the key expansion function creating the round keys. Next, it runs the AddRoundKey function starting off the encryption process. Then it starts looping SubBytes, ShiftRows, MixColumns and AddRoundKey for each round that there are keys. After the loop, it runs a final run of SubBytes, ShiftRows and AddRoundKey finishing off the encryption process and prints the encrypted grid before compleating the reverse and decrypting the data. To finish off it displays the decrypted grid which if the algorithm worked should match the plaintext grid.

Testing

AES 2001 Standard Test Vectors	My Program Under The Same Vectors

128-bit Plaintext: 011223344556677889 9aabbccddeeff Key: 000102030405060708090a0b0c0d0e0f Output: 69c4e0d86a7b0430d8cdb78070b4c55a	Select Option: 1.128Bit 2.192Bit 3.256Bit 4.384Bit 5.512Bit 6.1024Bit(Interlaced) 1 Hex Plaintext: 00112233445566778899aabbccddeeff Hex Key: 000102030405060708090a0b0c0d0e0f Raw Data: ------------- |00|44|88|CC| ------------- |11|55|99|DD| ------------- |22|66|AA|EE| ------------- |33|77|BB|FF| ------------- Key: 000102030405060708090A0B0C0D0E0F 128 Bit ------------- |69|6A|D8|70| ------------- |C4|7B|CD|B4| ------------- |E0|04|B7|C5| ------------- |D8|30|80|5A| ------------- Decrypt ------------- |00|44|88|CC| ------------- |11|55|99|DD| ------------- |22|66|AA|EE| ------------- |33|77|BB|FF| ------------- Test Results: Completed Successfully With Correct Answer.

192-bit Plaintext: 00112233445566778899aabbccddeeff Key: 000102030405060708090a0b0c0d0e0f 1011121314151617 Output: dda97ca4864cdfe06eaf70a0ec0d7191	Select Option: 1.128Bit 2.192Bit 3.256Bit 4.384Bit 5.512Bit 6.1024Bit(Interlaced) 2 Hex Plaintext: 00112233445566778899aabbccddeeff Hex Key: 000102030405060708090a0b0c0d0e0f1011121314151617 Raw Data: ------------- |00|44|88|CC| ------------- |11|55|99|DD| ------------- |22|66|AA|EE| ------------- |33|77|BB|FF| ------------- Key: 000102030405060708090A0B0C0D0E0F1011121314151617 192 Bit ------------- |DD|86|6E|EC| ------------- |A9|4C|AF|0D| ------------- |7C|DF|70|71| ------------- |A4|E0|A0|91| ------------- Decrypt ------------- |00|44|88|CC| ------------- |11|55|99|DD| ------------- |22|66|AA|EE| ------------- |33|77|BB|FF| ------------- Test Results: Completed Successfully With Correct Answer.
256-bit Plaintext: 00112233445566778899aabbccddeeff Key: Output: 8ea2b7ca516745bfeafc49904b496089	Select Option: 1.128Bit 2.192Bit 3.256Bit 4.384Bit 5.512Bit 6.1024Bit(Interlaced) 3 Hex Plaintext: 00112233445566778899aabbccddeeff Hex Key: 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f Raw Data: ------------- |00|44|88|CC| ------------- |11|55|99|DD| ------------- |22|66|AA|EE| ------------- |33|77|BB|FF| ------------- Key: 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F 256 Bit ------------- |8E|51|EA|4B| ------------- |A2|67|FC|49| ------------- |B7|45|49|60| ------------- |CA|BF|90|89| ------------- Decrypt ------------- |00|44|88|CC| ------------- |11|55|99|DD| ------------- |22|66|AA|EE| ------------- |33|77|BB|FF| ------------- Test Results: Completed Successfully With Correct Answer.

All the test vectors up to 256-bit completed successfully. This means that the new algorithm is identical to the original standard up to 256-bit. In each image it is important to note that the decrypt section is not just a copy of the input, instead, the program decrypts the encrypted block, this can be seen in the start function.

The following tests are not included in the AES standard and are just to demonstrate that the code executes correctly.

384-Bit

Plaintext: 00112233445566778899aabbccddeeff

Key:
472B4B6250655368566D597133743676397924422645294840 4D635166546A575468576D5A7134743777217A254 32A46

Output: 28c9b3d95292eb3426711b5a57d96e4d

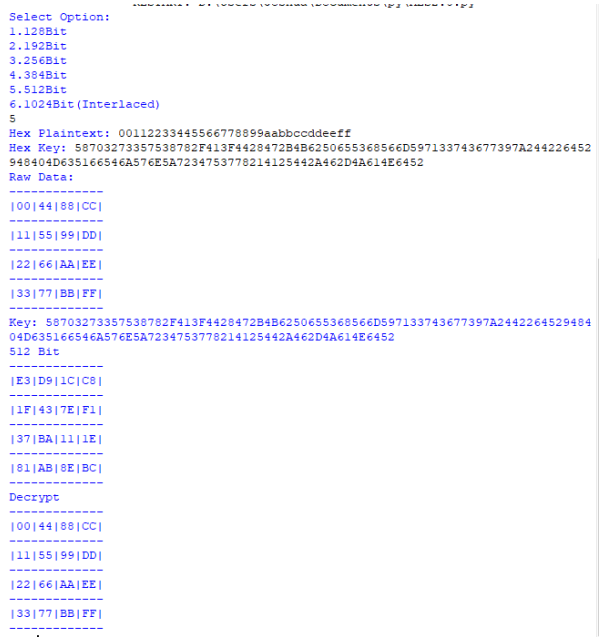
```
Select Option:
1.128Bit
2.192Bit
3.256Bit
4.384Bit
5.512Bit
6.1024Bit(Interlaced)
4
Hex Plaintext: 00112233445566778899aabbccddeeff
Hex Key: 472B4B6250655368566D597133743676397924422645294840 4D635166546A575468576
D5A7134743777217A25432A46
Raw Data:
-------------
|00|44|88|CC|
-------------
|11|55|99|DD|
-------------
|22|66|AA|EE|
-------------
|33|77|BB|FF|
-------------
Key: 472B4B6250655368566D597133743676397924422645294840 4D635166546A575468576D5A7
134743777217A25432A46
384 Bit
-------------
|28|52|26|57|
-------------
|C9|92|71|D9|
-------------
|B3|EB|1B|6E|
-------------
|D9|34|5A|4D|
-------------
Decrypt
-------------
|00|44|88|CC|
-------------
|11|55|99|DD|
-------------
|22|66|AA|EE|
-------------
|33|77|BB|FF|
-------------
```

512-Bit

Plaintext: 00112233445566778899aabbccddeeff

Key:
73367638792F423F4528482B4D6251655468576D5A7134743777217A24432646294A404E635266556A586E32723 57538782F413F442A472D4B6150645367566B

Output: e31f3781d943baab1c7e118ec8f11ebc



1024-Bit (Interlaced)

Plaintext: 00112233445566778899aabbccddeeff

Key:
58703273357538782F413F4428472B4B6250655368566D597133743677397A244226452948404D635166546A576E5A72347537782141254 42A462D4A614E6452

Key 2:
66556A586E32723575377821412544 2A472D4B6150645367566B59703373367639792F423F4528482B4D6251655468576D5A7134743777217A25432646294A40

Output: 4262ca29c4037ce29c07cb1445d0227d

```
Select Option:
1.128Bit
2.192Bit
3.256Bit
4.384Bit
5.512Bit
6.1024Bit(Interlaced)
6
Hex Plaintext: 00112233445566778899aabbccddeeff
Hex Key: 5870327335753878 2F413F4428472B4B6250655368566D597133743677397A244226452
948404D635166546A576E5A72347537782141254 42A462D4A614E6452
Hex Key 2: 66556A586E32723575377821412544 2A472D4B6150645367566B59703373676397 92
F423F4528482B4D6251655468576D5A7134743777217A25432646294A40
Raw Data:
-------------
|00|44|88|CC|
-------------
|11|55|99|DD|
-------------
|22|66|AA|EE|
-------------
|33|77|BB|FF|
-------------
Key: 5870327335753878 2F413F4428472B4B6250655368566D597133743677397A2442264529484
04D635166546A576E5A72347537782141254 42A462D4A614E6452
Key 2: 5870327335753878 2F413F4428472B4B6250655368566D597133743677397A24422645294
8404D635166546A576E5A72347537782141254 42A462D4A614E6452
2x 512 Bit
-------------
|42|C4|9C|45|
-------------
|62|03|07|D0|
-------------
|CA|7C|CB|22|
-------------
|29|E2|14|7D|
-------------
Decrypt
-------------
|00|44|88|CC|
-------------
|11|55|99|DD|
-------------
|22|66|AA|EE|
-------------
|33|77|BB|FF|
-------------
```

Critical Evaluations

Evaluating the project's success

When the project was started, the aim was to improve upon AES and now that the project is finished, the criteria has been met. However, even though the criteria that was set out has been achieved, the project could have been expanded to include more features, if more time had been spent planning at the beginning it is likely that a better algorithm could have been produced. That being said the increased bit size has increased the algorithms defence against brute force attacks by a factor of 1.5525180 × 10^231 times (2^1024/2^256), which means that it is statistically improbable that someone could brute force a 1024-bit key. The interlaced key is a great feature that adds a lot of confusion to the

algorithm. If there was a second chance to redo the project more time would be spent on planning as now the project is finished there are many ideas that would make the algorithm better that were discovered after the project's completion.

Project management

Now that the end of the project has been reached, it has been discovered project management is where I have the most room to grow. At the beginning of the project, there was very little research into ideas that could have been added to the project rather than focusing on getting a copy of AES to work on. Now that the project has been concluded I have realised that more time should have been allocated to research algorithms that are not in the university curriculum such as 3D encryption algorithms. More time should have also been put into planning the algorithm rather than coming in with a mental image of what the algorithm should look like. If the algorithm had been carefully planned out, there would have had been the opportunity to make features such as independent s-boxes. time management is another point that could be improved upon as well, although there were problems before and around Christmas, a plan should have been made for how to get back on track as well as more time dedicated to re-adjusting the design.

Research

Throughout the project, many modern encryption algorithms where researched that are taught in the course's curriculum as well as many found in online forum posts about AES. Research should have been conducted beyond AES and the other mainstream substitution permutation algorithms as they have many similar functions and features, these functions could also have been compared in further detail and more detailed notes could have been made.

Project Development

During the project, rapid application development methodology was used, now that the project is finished, RAD still appears to be the correct methodology for this project. However, the project would have turned out much better if more time were spent at the beginning, on research and design, as only 2 months were spent on planning, researching, and designing the algorithm. Rapid application development helped get a working application after Christmas and saved a lot of time. Some of the features found after Christmas where not able to be added to the algorithm without re-writing it, they would also prevent me from testing my code against the AES test vectors.

Experience Gained

Academically

I have learned a lot in academic skills in this project. Through my failure to research properly I have learned that my projects will be more successful if I take the time to research in greater detail and compare methods. I have also learned to look beyond methods taught in education, as when researching methods online it was found that methods could be added to my project. However, they are not taught in university as they are not being widely used.

Professionally

I have also learned professional skills during this project. Through failure to manage my time, I have learned that research and planning take long amounts of time and should not be cut short as it damages the deliverables in the project. I also learnt that when choosing my programming language to consider where my project is going and what I will need to add, for example, I can create a basic GUI in C# but not in python and now at the end of the project I wish I had given myself the option of adding a GUI.

What could be done differently next time?

Within this project, there where multiple ideas for increasing the security of AES that could not be added due to either my lack of the skills required to implement the feature or a lack of time. Hears some of the ideas that if the project were restarted could be added:

3D data block

Idea from (Jorge Nakagara Jr., 2008). One idea that could be added but would a complete rebuild of AES from the start is instead of putting the data in a 4x4 array, put it in a 3D array with 4x4x4. This would require every function to

work in another axis. However, it would increase the effectiveness of the shiftrows and mixcolumns functions as data can move in another direction increasing the number of spaces one cell of data can move to and thus increasing the confusion and diffusion these functions cause. In addition to this, a new function to rotate parts of the cube could be added like a Rubix cube. The strength of this idea is that it would increase the complexity of the encryption making it harder to crack, the weakness of this idea is that it would also increase the time taken to compute and so would make the algorithm less useful for everyday use and should only be used for confidential high-risk data that is not often accessed. If I were to use this method in the future, it would have to developed in a faster coding language like C++ and precautions to optimise the code as much as possible to reduce the time taken to encrypt and decrypt would have to be taken, this is because there is 4x as much data being processed in a 4x4x4 block compared to a 4x4 block and so would take 4x longer to compute.

Key dependent S-boxes

*"The use of key-dependent substitution matrices (S-BOXes) is one of the commonly used methods to change the characteristics of a cryptographic algorithm." (Stoianov N, 2011).*When research into Twofish was conducted after writing the code It was noticed that Twofish uses key-dependent S-boxes. Key dependent s-boxes work by at the start of the algorithm, during the key expansion stage, the algorithm generates the s-box based of the key. This means that the Subbytes function which on its own can easily be decrypted without a key is instead made by the key, because of this it becomes very difficult to bypass without the key just like the AddRoundKey function. This function can no longer be added due to time concentrates. This would slightly increase the amount of time taken to compute however it would be milliseconds and would only have an impact on large files.

GUI (graphical user interface)

A GUI is used to help users who are not knowledgeable about the computing field and how to use a console. I have next to no experience in front end development and would have to do extensive research and teach myself how to create one, for this reason, it was decided at the beginning of this project not to bother with a GUI however if there was another year to work on this project, I would teach myself the skills required to add this feature. (*Computer Hope, 2021*)

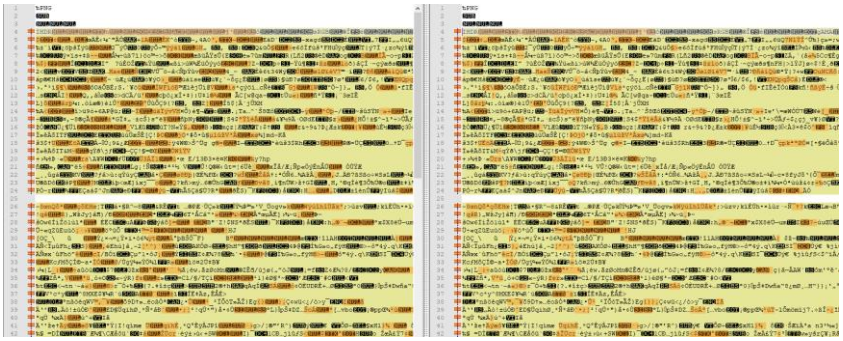
File handling

There was an attempt to add file handling at the beginning of this project however there were problems when trying to convert a file to binary then formatting it into 128-bit blocks, so it was decided that it would be added later on. When this was tried this in January, data was added into cells and back out however an error that could not be found corrupted the file by up to 20%.

```
import binascii
import math
import tkinter as tk
from tkinter import filedialog
blocks = []

def display(data): # print the 4x4 display to screen (debugging)
    print("-------------")
    for i in range(0,4):
        print("|" + data[i][0] + "|" + data[i][1] + "|" + data[i][2] + "|" + data[i][3] + "|")
        print("-------------")

filename = 'custom.PNG'
with open(filename, 'rb') as f:
    content = f.read()
#hexdata = binascii.hexlify(content)
hexdata = bytes.hex(content)
final_block =[["00","00","00","00"],["00","00","00","00"],["00","00","00","00"],["00","00","00","00"]]
temp_block=["00","00","00","00","00","00","00","00","00","00","00","00","00","00","00","00"]
c=0
for i in range(0,len(hexdata),2):
    temp_block[c] = hexdata[i] + hexdata[i+1]
    if (c==15):
        c2=0
        for x in range(0,4):
            for y in range(0,4):
                final_block[x][y] = temp_block[c2]
                c2 +=1
        blocks.append(final_block)
        temp_block=["00","00","00","00","00","00","00","00","00","00","00","00","00","00","00","00"]
        final_block =[["00","00","00","00"],["00","00","00","00"],["00","00","00","00"],["00","00","00","00"]]
        c=0
    c+=1

undo = ""
for i in range(0,len(blocks)):
    for x in range(0,4):
        for y in range(0,4):
            undo += blocks[i][x][y]

with open('test.PNG', 'wb') as f:
    for i in range(0,len(undo),2):
        towrite = undo[i] + undo[i+1]
        f.write(binascii.unhexlify(towrite))
```

File handling test code

This code was developed separately from the main AES code to avoid causing errors. The code runs however for some reason the output is corrupted.



Comparison between the original picture and one that has gone through the file handling test code, discrepancies highlighted in orange.

User input cleaning

User input cleaning is where a programmer wites code to prevent the users entering values that would cause error or cause a buffer overflow. "*For example, if I ask the user to enter either "hello" or "goodbye", and they type something else, I need it to tell the user it's wrong and ask again*". User input handling has not been added as there is no GUI or file handling, and the code can only encrypt one 128-bit block every time it is launched, this is because the program is only written to show that the improved algorithm works. As the program was made to show the algorithm it was decided not to include any input cleaning as it would just take time and testing and yield no benefit to the encryption. (*User2869231, 2013*)

C# / C++

If a second attempt at the project the algorithm would be built with the above features, for the second attempt C# or C++ would be used as the programming languages as they make creating GUI's and File handling a lot easier. The decision was made python for the project this time because of how quick it is to develop programs with, however for a full version with GUI's it would be preferable to take the time and use C# or C++. These languages also have the

added benefit of being slightly faster "*Python takes 25 times more time to run the same algorithm compared to C++*". (*Tamimi N, 2020*) (*Microsoft, 2019*)

Optimization

The program could be optimised to support parallel processing on some functions, for example, sub bytes could have each cell ran at the same time, for shift rows each row could be processed at the same time, mixcoulums could be run on every cell and add round key could be run on every cell simultaneously. "*Parallel processing is a method in computing of running two or more processors (CPUs) to handle separate parts of an overall task. Breaking up different parts of a task among multiple processors will help reduce the amount of time to run a program*". (*Tech Target Contributor, no date)*

Keys for individual blocks

From Stack Exchange a fourm thread was found talking about if AES used a key per data block or not "*If you have 4 16-byte blocks of plaintext to encrypt does each block have a different 128-bit original key? Or do they all use the same?*". This gave the idea of using a key for each block of data. If a GUI were created along with file handling for the algorithm a key generator would also be made for users to store their keys digitally. This could create a massive key which is split into a key block for each plaintext block in the document. This would give massive variation on each data block and would be near impossible to decrypt. If a user tried to encrypt a 1Mb document (actually 999,936 bit document) there would be 7812 different keys if the user encrypted them with 1024 bit interlaced mode the amount of combinations to solve to get the whole document would be 2^7999488 compared to the normal 1024^256 this is a massive jump in the time taken to brute force, however the key would be 8Gb long (actually 7,999,488 bit) 1000x the size of the document encrypted. The number of combinations is 6.89 x10^2408085 making the algorithm statistically improbable to brute force. (*Green, 2018*)

There are many practical applications of AES, for example this method will be extremely slow and require 1000 times more resources the amount of security against brute force attacks is near impenetrable, due to this its use would be best suited for highly secret documents such as those within the military. This feature could be used to secure Top classified documents such as nuclear weapons designs. One military application of AES currently in use is "*AES-128 is used for secret (unclassified) information and AES-256 for top secret (classified) information.". (NordPass, 2020)*

Combination of the above

An idea that seems interesting is to combine the above ideas along with the current build of AES. The project would be stated from scratch and a few weeks would be spent looking for more ideas to add the above list. A month would then be spent on planning the design of the algorithm and how the different ideas can combine without forfeiting any security. The algorithm would still be loosely based off AES however incorporating the 3D aspect will leave the algorithm looking very different to AES source code. The code could be optimised much more thoroughly to make the code execute faster. It would have to have a front-end GUI and file handling to make the application more user friendly, of course adding input cleaning to prevent errors. Keys for individual blocks would be added however it would be made optional as the key would need to be a document as its size would make it impossible to memorize, it would also need to be randomly generated as the user would need to input a lot of data to reach the required key size. The randomly generated key could use user input as a way of making the data truly random such as using the mouse in some way.

Conclusion

In conclusion, the project was successful in achieving the aim that was set, to improve upon AES to increase computational complexity and thus improving the security of the algorithm. Within this project the features added to improve security where an increase of the key size as well as an interlaced dual key system. The key size was originally 256-bit witch was increased to 512-bit as well as 1024-bit with the interlaced key system. Even though the

key size was increased it can be further increased indefinitely. However, increasing the key size further is ineffective as 1024-bit is already considered excessive and extreme. (*Legorooj, 2019*)

From face value the project was successful and there are many features that could have been added to further increase the usability and effectiveness of the application. As the project is now users who are not experienced with encryption and navigating a command line interface are unable to use the software. One of the functions mentioned in the critical evaluation is a GUI, this would allow users with no experience the ability to use the encryption algorithm. The GUI could be paired with user input handling as well as file handling to make the algorithm work for all users on full documents. With these changes the program would no longer be a simple test but instead an application that can be used in day-to-day life to secure user documents. (*Computer Hope, 2021*)

Although there were not enough features added to the algorithm, the reason some of the features where not added is that there was not enough research conducted at the beginning of the project and by the time the features were researched after Christmas it was too late into the project and would require a rebuild of most of the program. More time should also have been spent planning in the project, as more features could have been added such as a GUI and the program could have been written in C# or C++. The changes made to AES have increased confusion within the algorithm and strengthened it against brute force attacks. If there was a second chance at this project, more of the features listed above would be added to the current algorithm for the base, however it would be recoded in another programming language.

The official AES documentation was written in 2001 20 years ago, which could indicate the need for reviewing and the revision of the document. However, AES including 128-bit has not been cracked during its lifespan. The reason for this project is to create an algorithm secure enough to last further into the future. To conclude this project has increased the time taken to brute force the algorithm to last in a future where technologies such as quantum computers may crack regular AES. (*Federal Information Processing Standards Publication, 2001*)

Bibliography

BCS (No date) *BCS Code of Conduct.* Available at: https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/ (Accessed: 23/04/2021)

British Computer Society (2015) *BCS, THE CHARTERED INSTITUTE FOR IT*

Britannica (no date) *Cipher*. https://www.britannica.com/topic/cipher (Accessed: 05/05/2021)

CODE OF CONDUCT FOR BCS MEMBERS. Version 5 Available at: https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf (Accessed: 1 April 2021).

Compute Hope (2021) *GUI* Available at: https://www.computerhope.com/jargon/g/gui.htm (Accessed 03/04/2021)

Conway L (2020) *Blockchain Explained* Available at: https://www.investopedia.com/terms/b/blockchain.asp (Accessed: 13/04/2021)

Cryptography Fandom (no date) *Rijndael mix columns.* Available at: https://cryptography.fandom.com/wiki/Rijndael_mix_columns (Accessed 05/05/2021)

Educative (No date) *What are Substitution-Permutation networks?* Available at: https://www.educative.io/edpresso/what-are-substitution-permutation-networks (Accessed: 13/04/2021)

Federal Information Processing Standards Publication (2001) *Advanced Encryption Standard (AES) (FIPS PUB 197)*. Available at: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (Accessed: 30 January 2021).

Gatliff B (2003) *Encrypting data with the Blowfish algorithm.* Available at: https://www.embedded.com/encrypting-data-with-the-blowfish-algorithm/#:~:text=Blowfish%20is%20a%20symmetric%20encryption,blocks%20during%20encryption%20and%20decryption. (Accessed: 12/04/2021)

GeeksforGeeks (2020) *What is RC4 Encryption?* Available at: https://www.geeksforgeeks.org/what-is-rc4-encryption/ (Accessed: 24/04/2021)

GeeksforGeeks (2021) *Caesar Cipher in Cryptography.* Available at: https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/ (Accessed: 13/04/2021)

Green (2018) *Does AES use different keys for each block of plaintext?* Available at: https://crypto.stackexchange.com/questions/61218/does-aes-use-different-keys-for-each-block-of-plaintext (Accessed 06/05/2021)

Guru99 (No date) *Python vs C++: What's the Difference?* Available at: https://www.guru99.com/python-vs-c-plus-plus.html#:~:text=KEY%20DIFFERENCES%3A,C%2B%2B%20is%20faster%20than%20Python (Accessed: 17/03/2021)

Jorge Nakagara Jr. (2008) *3D: A Three-Dimensional Block Cipher.* Available at: https://link.springer.com/chapter/10.1007/978-3-540-89641-8_18 (Accessed: 1 April 2021).

Josh Lake (2020) *What is AES encryption and how does it work?* Available at: https://www.comparitech.com/blog/information-security/what-is-aes-encryption/ (Accessed: 1 April 2021).

KAVALIRO (2014) *AES Example.* Available at: https://kavaliro.com/wp-content/uploads/2014/03/AES.pdf (Accessed: 1 April 2021).

Legorooj (2019) *AES with bigger keys more secure?* Available at: https://crypto.stackexchange.com/questions/72129/aes-with-bigger-keys-more-secure (Accessed 04/05/2021)

Microsoft (2019) Create a Windows Forms app in Visual Studio with C# Available at: https://docs.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2019 (03/04/2021)

Moserware (2009) *A Stick Figure Guide to the Advanced Encryption Standard (AES)* Available at: http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html (Accessed: 1 April 2021).

Mohit Arora (2012) *How secure is AES against brute force attacks?* Available at: https://www.eetimes.com/how-secure-is-aes-against-brute-force-attacks/ (Accessed: 7 April 2021)

Murphy, S. (2002) *Key-dependent S-boxes, Differential Crypanalysis, and Twofish*. Available at: https://www.isg.rhul.ac.uk/~sean/tf_dckdsb.pdf (Accessed: 1 April 2021).

Niklas-heer (2018) *speed-comparison.* Available at: https://github.com/niklas-heer/speed-comparison (Accessed: 1 April 2021).

NIST (2017) *Update to Current Use and Deprecation of TDEA.* Available at: https://csrc.nist.gov/News/2017/Update-to-Current-Use-and-Deprecation-of-TDEA (Accessed: 05/05/2021)

NordPass (2020) *Military-Grade Encryption Explained.* Available at: https://nordpass.com/blog/military-grade-encryption-explained/#:~:text=The%20US%20government%20specifies%20that,top%20secret%20(classified)%20information.&text=As%20AES%20is%20used%20by,military%2Dgrade%E2%80%9D%20seemed%20suitable. (Accessed: 01/05/2021)

Parahar M (2020) *Difference between Private Key and Public Key* Available at: https://www.tutorialspoint.com/difference-between-private-key-and-public-key#:~:text=Algorithm-,Private%20Key%20is%20used%20to%20both%20encrypt%20and%20decrypt%20the,is%20used%20and%20is%20shared.&text=The%20public%20key%20is%20free,key%20is%20kept%20secret%20only. (Accessed 21/04/2021)

PCMag (no date) *XOR* Available at: https://www.pcmag.com/encyclopedia/term/xor (Accessed: 11/04/2021)

Pedamkar P (no date) *Digital Signature Algorithm.* Available at: https://www.educba.com/digital-signature-algorithm/ (Accessed 06/05/2021)

Peter Loshin (2016) *Retiring obsolete SHA-1 and RC4 cryptographic algorithms, SSLv3 protocol* Available at: https://searchsecurity.techtarget.com/news/450297247/Retiring-obsolete-SHA-1-and-RC4-cryptographic-algorithms-SSLv3-protocol (Accessed: 05/05/2021)

RF Wireless World(no date) *Advantages of AES | disadvantages of AES.* Available at: https://www.rfwireless-world.com/Terminology/Advantages-and-disadvantages-of-AES.html (Accessed: 7 April 2021)

Sage (no date) *S-Boxes and Their Algebraic Representations.* Available at: https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/sbox.html (Accessed: 11/04/2021)

Satish C J (2020) *AES III - Advanced Encryption Standard - Introduction , Key Expansion in AES Cyber Security CSE4003*. 26 August. Available at: https://www.youtube.com/watch?v=w4aWIVhcUyo (Accessed: 1 April 2021).

Satish C J (2020) *AES IV - Advanced Encryption Standard - Encryption and Decryption - Cyber Security CSE4003*. 26 August. Available at: https://www.youtube.com/watch?v=5PHMbGr8eOA (Accessed: 1 April 2021).

Simply Explained (2017) *How does a blockchain work - Simply Explained* Available at: https://www.youtube.com/watch?v=SSo_EIwHSd4 (Accessed: 10/05/2021)

Singh A (2019) *What Is Rapid Application Development (RAD)?* Available at: https://blog.capterra.com/what-is-rapid-application-development/#:~:text=Rapid%20Application%20Development%20(RAD)%20is,strict%20planning%20and%20requirements%20recording. (Accessed: 27/03/2021)

SoolarWindsMSP (2019) Understanding AES 256 Encryption Available at: https://www.solarwindsmsp.com/blog/aes-256-encryption-algorithm#:~:text=AES%20brings%20additional%20security%20because,harder%20to%20break%20the%20encryption. (Accessed: 1 April 2021).

Stoianov N (2011) *One Approach of Using Key-Dependent S-BOXes in AES* Available at: https://link.springer.com/chapter/10.1007/978-3-642-21512-4_38 (Accessed 10/05/2021)

Tamimi N (2020) *How Fast Is C++ Compared to Python?* Available at: https://towardsdatascience.com/how-fast-is-c-compared-to-python-978f18f474c7 (Accessed: 09/05/2021)

Tech Target Contributor (no date) *parallel processing.* Available at: https://searchdatacenter.techtarget.com/definition/parallel-processing#:~:text=Parallel%20processing%20is%20a%20method,time%20to%20run%20a%20program. (Accessed 03/05/2021)

Townsend Security (2020) *AES vs. DES Encryption: Why Advanced Encryption Standard (AES) has replaced DES, 3DES and TDEA.* Available at: https://www.precisely.com/blog/data-security/aes-vs-des-encryption-standard-3des-tdea (Accessed: 05/05/2021)

Thomas, R. (2020) *Advanced Encryption Standard (AES): What It Is and How It Works.* Available at: https://securityboulevard.com/2020/04/advanced-encryption-standard-aes-what-it-is-and-how-it-works/ (Accessed 10/05/2021)

User2869231(2013) *How to do User Input Error Handling in Python?* Available at: https://stackoverflow.com/questions/19408087/how-to-do-user-input-error-handling-in-python (Accessed: 04/04/2021)

Yackel, R. (2020) *When to Use Symmetric Encryption vs. Asymmetric Encryption.* Available at: https://blog.keyfactor.com/symmetric-vs-asymmetric-encryption (Accessed: 05/05/2021)