Implementation and Performance study of Advanced Encryption Standard

Nirajan Koirala
Electrical and Computer Engineering Department
Villanova University
nkoirala@villanova.edu

Abstract— Various symmetrical and asymmetrical encryption techniques are used to ensure security in the field of computer networking. This work presents a study of a symmetric-key encryption technique, Advanced Encryption Standard, and its underlying implementation. First, we provide a brief Mathematical basis necessary for understanding the specifications of the algorithm and then discuss the main steps used in the encryption process. Implementation of AES is done in different modes and the performance level of the algorithm is measured for various kinds of inputs. Finally, we discuss some other aspects of AES like its strength and its uses besides encryption.

***Index Terms*—AES, CBC, CFB, OFB, CTR.**

I. Introduction

Advanced Encryption Standard (AES), also known as Rijndael is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. It supersedes Data Encryption Standard (DES) which was published in 1977 [22]. DES, which uses a key of 56-bit size, was found to be vulnerable to brute-force attacks and although 3-DES was designed to overcome this key-size vulnerability of DES, it was slower in performance. AES was chosen as the new standard due to its higher security and performance levels. According to Demon et al., the major criteria taken into account while designing AES were resistance against all known attacks, good performance on a wide range of platforms, and design simplicity [5]. AES is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in block-sizes of 128 bits.

II. History

AES was primarily developed to fulfill the security needs of the US government around the 2000s. The federal agencies previously relied on DES as the primary encryption algorithm which was created by IBM with a 56-bit symmetric-key block cipher design [22]. During the 1990s, several DES challenges were hosted by RSA Security and DES was shown to be breakable within a matter of days if not hours. In 1999, distributed.net and Deep Crack were able to crack DES in less than 23 hours making it evidently clear that DES with a 56-bit key could no longer provide sufficient security [20].

In January 1997, NIST issued a public call for candidates to replace the aging DES, which resulted in 15 viable submissions from 12 countries [21]. The scientific community lauded this open nature of the process, which subjected each of the encryption algorithms to public security and also by doing so, the government could be sure that no system had a backdoor, and the chances of identifying and fixing flaws were maximized [22]. The submissions were scrutinized in a first-round resulting in 5 final candidates for AES which included MARS, RC6 (Rivest Cipher), Serpent, Twofish, and Rijndael. These finalist algorithms received further analysis during a second, more in-depth review period. Finally, in October 2000, NIST announced Rijndael, an encryption technique created by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, to be the new standard, or the Advanced Encryption Standard (AES) [23]. Rijndael was chosen because of its high-security, performance, efficiency, implementability, and flexibility.

III. Basic Mathematical Preliminaries

All the mathematical operations in AES are performed in the Galois Field GF(2^8) which contains 2^8, or 256, elements. Any byte value during the encryption process can be mapped to exactly one element of GF(2^8). Working in a finite field like GF(2^8) has many benefits [26]. Firstly, the numbers stay small even after big operations like multiplication and division. We also make sure that inverses exist for each operation during decryption. Additionally, rich operations like transpose in row shift and matrix operations during mix columns and creation of substitution box (S-box) are possible [26]. Most common operations like addition and multiplication are discussed in this section and for more in-depth information, one can refer to Benvenuto et al. [6]. All the operations and results in this section are verified using an online AES implementation provided by Styer [10].

A common representation of the elements of GF(2^8) is a polynomial of degree seven with coefficients as either 0 or 1. For instance, a byte b consisting of bits b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 is mapped to GF(2^8) as the polynomial,

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0.$$

Using this mechanism, the byte with hexadecimal value 42 (binary 00101010) corresponds with the following polynomial,

$$x^5 + x^3 + x.$$

A. Addition Operation in GF(2^8)

The addition of two elements in GF(2^8) results in a polynomial with coefficients that are produced by sum *modulo* 2 of the coefficients of the two terms. This operation corresponds with a standard bitwise XOR operation.

For instance, '42' + '37' = '0F', or with the polynomial notation we have,

$$(x^5 + x^3 + x) + (x^5 + x^2 + 1) = x^3 + x^2 + x + 1$$

In binary we have,

$$00101010 + 00100101 = 00001111$$

B. Multiplication Operation in GF(2^8)

Multiplication in GF(2^8) requires more tedious work. This operation corresponds with a multiplication of polynomials modulo an irreducible binary polynomial of degree 8. In AES, this polynomial is denoted by m(x) and where,

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

or '11B' in hexadecimal.

For instance, '41' • '84' = 'D2', or with the polynomial notation we have,

$$(x^6 + 1)(x^7 + x^2) = x^{13} + x^8 + x^7 + x^2$$

Then we compute, it's modulo with m(x),

$$x^{13} + x^8 + x^7 + x^2 \text{ modulo } x^8 + x^4 + x^3 + x + 1$$
$$= x^7 + x^6 + x^4 + x.$$

IV. Encryption Procedure

In this section, we discuss the steps involved in the encryption process of AES. AES is a 128-bit block cipher that supports independent key sizes of 128, 192, or 256 bits. The algorithm performs multiple rounds of encryption and the number of rounds depends on the chosen key length. Using key sizes of 128, 192, and 256 bits requires 10, 12, and 14 rounds respectively.

We discuss the steps used in the algorithm for the case when the key size is 128 bits. The steps can be broken down in the following manner:

i. Key-expansion
ii. Initial Round Key addition:
   a. AddRoundKey
iii. Rounds (9):
   a. SubBytes
   b. ShiftRows
   c. MixColumns
   d. AddRoundKey
iv. Final Round (1):
   a. SubBytes
   b. ShiftRows
   c. AddRoundKey

The algorithm consists of an initial XOR step, 9 rounds of transformation, and one additional round performed at the end with MixColumns step omitted. The input matrix to each round is called a state and each round consists of layers. The encryption procedure can be visualized using figure 1.

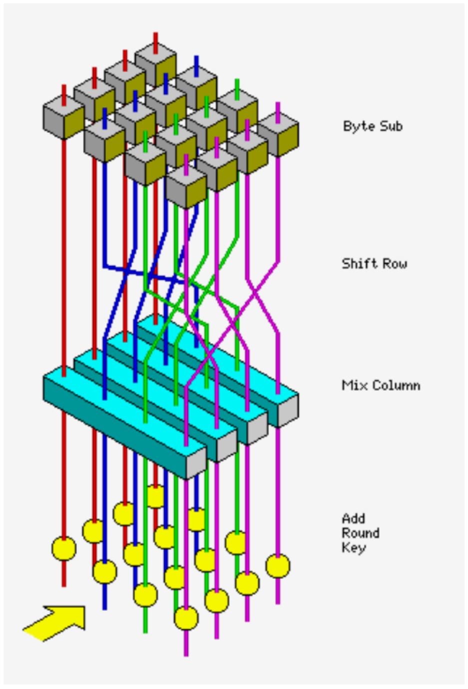

Figure 1: A single round of AES Encryption (Figure Courtesy of Carpenter [3])

A. Key-expansion

A cipher key is expanded for use in different rounds of the algorithm in this stage. Each round of AES uses a different key (subkey). These subkeys are derived from the Cipher Key by means of the Key Schedule and it differs for different key sizes. The number of subkeys also depends on the key size and the subkeys for key lengths 128, 192, and 256 bits are 11, 13, and 15 respectively. These subkeys are derived recursively from the original 128/192/256 bit key in the Key Schedule. After the Cipher Key is expanded into an Expanded key, it is stored in a linear array of 4-byte words which can be used for accessing the subkeys for each round.

In the case we are following, a 128-bit key would generate 11 subkeys. The first subkey is always the original AES key and the remaining 10 subkeys are computed using the key schedule algorithm. For further details on how the subkeys are generated in the key schedule, one can refer to [5].

B. Initial Round Key Addition

For an input and key size of 16 bytes, the State and Cipher key can be represented using 4x4 matrices. The input and key are placed in a matrix in column major order in the first step.

For instance, a byte message '11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26' can be represented as the matrix M1.

```
11 15 19 23
12 16 20 24
13 17 21 25
14 18 22 26
```

M1: First State

A cipher key '0f 15 71 c9 47 d9 e8 59 0c b7 ad d6 af 7f 67 98' can be represented as the matrix M2.

```
0f 47 0c af
15 d9 b7 7f
71 e8 ad 67
c9 59 d6 98
```

M2: Cipher Key

After the subkeys are generated, the subkeys are XORed with states and the resulting state is passed to the consecutive rounds. In the Initial Round Key addition layer, the subkey is equal to the cipher key. The subkey is added in GF(2^8) or XORed with the state.

For instance, XOR of the first state (M1) and subkey (M2) after the AddRoundKey step would result in matrix M3.

```
1e 52 15 8c
07 cf 97 5b
62 ff 8c 42
dd 41 f4 be
```

M3: State after Initial AddRoundKey

C. Layers in the Rounds

Each round of AES consists of different layers such as SubBytes, ShiftRows, MixColumns and AddRoundKey except the final round which only consists SubBytes, ShiftRows and AddRoundKey layers.

1. SubBytes

In the SubBytes layer, each individual byte of the state is replaced using a S-box (table 1, figure 2). This table is invertible so that during decryption, S-box can be uniquely reversed.

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

Table 1: S-Box (Table Courtesy of Hansen et al. [16])

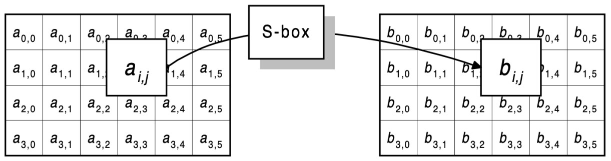
S-box
$a_{i,j}$
$b_{i,j}$
$a_{0,0}$ $a_{0,1}$ $a_{0,4}$ $a_{0,5}$
$a_{1,0}$ $a_{1,1}$ $a_{1,4}$ $a_{1,5}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$
$b_{0,0}$ $b_{0,1}$ $b_{0,4}$ $b_{0,5}$
$b_{1,0}$ $b_{1,1}$ $b_{1,4}$ $b_{1,5}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{2,4}$ $b_{2,5}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$ $b_{3,4}$ $b_{3,5}$

Figure 2: SubBytes Operation (Figure Courtesy of Daemen et al. [5])

After the SubBytes step, matrix M3 would be converted into M4.

```
72 00 59 64
c5 8a 88 39
aa 16 64 2c
c1 83 bf ae
```

M4: State after SubBytes

2. ShiftRows and MixColumns

The next two layers, ShiftRows and MixColumns, are together called the diffusion layers. ShiftRows performs permutation of the data on a byte level where the rows of the state matrix are shifted cyclically [1]. The first row contains no shift, the second row is shifted one position to the left, the third is shifted two positions to the left and the fourth row is shifted three positions to the left in the resulting state matrix after ShiftRows. After execution of the ShiftRows layer on M4, the output matrix is M5.

```
72 00 59 64
8a 88 39 c5
64 2c aa 16
ae c1 83 bf
```

M5: State after ShiftRows

The next sublayer in the diffusion layer, MixColumns, performs matrix operation on the state which combines blocks of four bytes. Each column of the state is multiplied by the polynomial,

c(x) = ‘03’ x + ‘01’ x + ‘01’ x + ‘02’

This polynomial is coprime to $x^4 + 1$ and therefore invertible while decryption [5]. The operation is equivalent to multiplying each column of the state matrix by the following matrix,

```
02 03 01 01
01 02 03 01
01 01 02 03
03 01 01 02
```

and it results in linear transformation of the original matrix (figure 3).

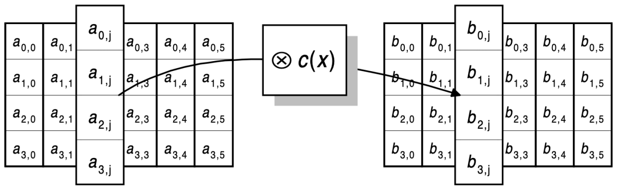

Figure 3: MixColumns Operation (Figure Courtesy of Daemen et al. [5])

The resulting matrix after applying MixColumns operation on matrix M5 is M6.

```
ab 6e d0 35
7f be 4d 70
d9 88 b1 57
3f 3d 65 1a
```

M6: State after MixColumns

3. AddRoundKey

In this stage the subkey generated during the Key-expansion process is used to XOR it with the state matrix at the current stage. This key is also termed as the round key as it is used in the current round. In the encryption process we are following, the subkey for round 2 produced during the Key-expansion using key-schedule is M7.

```
dc 9b 97 38
90 49 fe 81
37 df 72 15
b0 e9 3f a7
```

M7: Subkey for Round 2

This subkey, M7, is again XORed with the current state, M6, in this case, and the encryption process carries into next rounds until the final round is reached.

D. Final Round

Layers discussed in section C are performed for 9 rounds until we reach the final round of encryption. The last round of encryption omits the MixColumns layer and only performs SubBytes, ShiftRows and AddRoundKey operations. According to Daemon et al., this omission has no security implications and it makes the cipher and it’s inverse more similar in structure [5].

Matrix M1 would be transformed into M8 using the cipher key M2 after all the encryption rounds are completed.

```
62 ed 90 1b
cb 3e 17 93
db 55 6a c2
80 9f 31 50
```

M8: Final State (Ciphertext)

DECRYPTION STAGE

The decryption of a cipher in AES is carried out using the inverse function of each layer in the encryption process. We go in the bottom-up direction, undoing each step by applying its inverse transformation. The decryption mechanism for a cipher encrypted using a 128-bit key size is shown in figure 4.

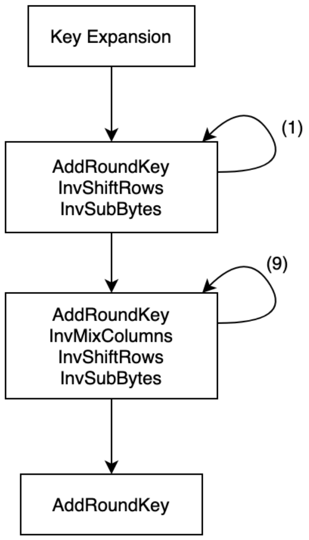

Figure 4: Decryption in AES (128-bit key)

V. MODES OF AES

Various standard modes of AES encryption are briefly discussed in this section. Electronic CodeBook (ECB) mode of encryption was omitted in this study as ECB is considered to be a very weak mode of AES. According to [6], the encryption in this mode is deterministic, and identical plaintext blocks produce identical ciphertext blocks. Moreover, the ordering of the ciphertext blocks can be easily changed by the attackers as there is no mechanism present to identify their ordering on the receiver side.

A. Cipher Block Chaining Mode (CBC)

CBC encryption mode was invented by IBM in 1976 [4]. This mode starts by XORing the first plaintext block with a random initialization vector (IV). Then, encryption is applied to the resulting block using a key K. Each subsequent plaintext block is XORed with the previous ciphertext block before performing encryption with K. CBC mode during encryption can be visualized in figure 5. Since every new block of a cipher depends on the previous block, this mode of AES cannot be parallelized.

The decryption of this mode is performed by XORing the output block obtained from the decryption algorithm using K to the previous ciphertext block. Since all the ciphertext blocks are known in hand, the decryption process can be parallelized in CBC mode of AES [4].

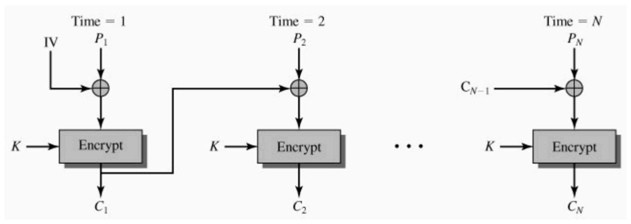

Figure 5: Encryption in CBC mode (Figure Courtesy of Blazhevski et al. [2]

B. Cipher FeedBack Mode (CFB)

CFB mode is a class of stream ciphers. The encryption mechanism of CFB is similar to CBC's as a chain of events can be observed in this mode as well (figure 6). First, an IV is encrypted using a key K. The resulting block of this operation is XORed with the plaintext block and a ciphertext block is generated. For the next round of encryption, ciphertext block from the previous round is encrypted with K and then XORed with the next plaintext block. This kind of encryption of the ciphertext doesn't affect the security of the algorithm and the same encryption algorithm can be used for the decryption process as well [4].

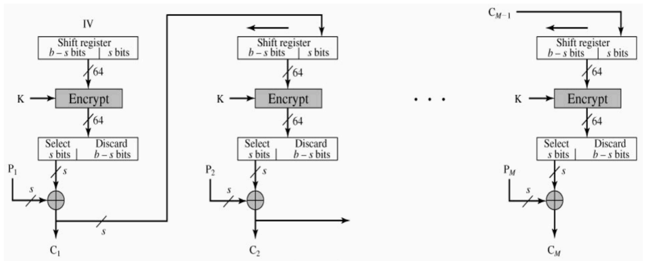

Figure 6: Encryption in CFB mode (Figure Courtesy of Blazhevski et al. [2])

C. Output FeedBack Mode (OFB)

OFB mode is also a class of stream ciphers and in a very similar fashion to CFB, the generated pseudo-random stream is XORed with the plaintext to generate the ciphertext. The encryption and decryption processes are handled by the same algorithm and none of them are parallelizable. OCB mode of AES is shown in figure 7.

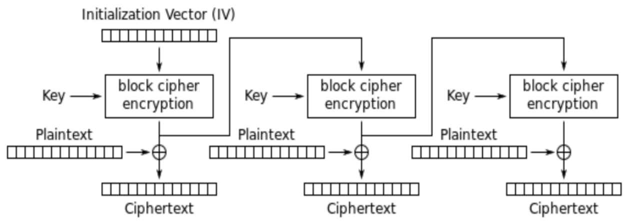

Figure 7: Encryption in OFB mode (Figure Courtesy of [9])

D. Counter Mode (CTR)

CTR mode generates a pseudo-random stream of data that may or may not depend on the plaintext. It also belongs to a class of stream ciphers. A counter block is used as an IV to the encryption algorithm with key K and it increments on every subsequent encryption (figure 8). The values of counters are independent of the output from the previous round so an error is not propagated in this mode. Similar to CFB and OFB, CTR also uses the same algorithm for encryption and decryption. Unlike previous modes, encryption and decryption both are parallelizable in this mode and thus this mode should be able to provide greater performance if implemented correctly using parallelization [11].

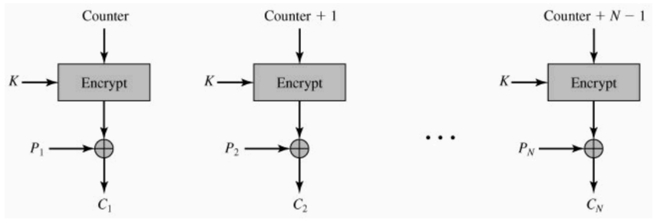

Figure 8: Encryption in CTR mode (Figure Courtesy of Blazhevski et al. [2])

Implementation of AES in CTR mode was done using Pyaes [8] in this study. This implementation found in [8] is not designed to be parallelizable and thus a very poor performance was observed while encryption. For comparison, a text file containing the United-States constitution (49KB) was encrypted and it took about 0.25 seconds in Pyaes CTR mode while the encryption of the same file took about 0.005 seconds in OFB and CBC mode using PyCryptoDome [7] implementation. The performance figures of this mode were thus omitted as it was not fair to compare modes of AES implemented by different vendors.

VI. Experimental Results

In this section, we discuss the implementation and experimental details of AES in the modes discussed in the earlier section. The implementation of the algorithm from scratch was not possible given the time and scope of this project. The algorithm was implemented in Python using PyCryptoDome Library [7] and an open-source implementation of the algorithm provided by Pyaes [8].

The experiments were run on an Apple MacBook Pro with a 2.5 GHz Quad-Core Intel Core i7 processor and a 16 GB 1600 MHz DDR3 RAM. Three different types of data files were used for encryption and the size of the total files present in the data folder was made to match each other. Types of data used:

- Images: JPG, PNG format (1.13GB)
- Audio Files: AAC and MP3 format (1.13 GB)
- Video Files: MOV and MP4 format (1.12GB)

Finally, all the scripts used for encryption and decryption are made available using the following link:
https://github.com/nirajank07/AES_encryption

The amount of time taken by the three modes of AES to encrypt and decrypt the same sized files can be observed in table 2.

AES Mode	**Performance (seconds)**		
	File Type	***Encryption***	***Decryption***
CBC	Image	6.50	5.27
	Audio	5.05	4.97
	Video	7.28	7.01
	All three	20.43	20.13
CFB	Image	41.93	37.53
	Audio	39.91	38.14
	Video	39.75	39.05
	All three	125.40	122.45
OFB	Image	6.12	5.45
	Audio	5.89	5.65
	Video	7.18	6.86
	All three	20.97	20.56

Table 2: Time taken by different modes of AES

A noticeable trend seen in table 2 is that the decryption is always faster (2-10%) than the encryption process in all of the modes. The audio files are encrypted and decrypted slightly faster than image files followed by video files. Time taken by the CBC and OFB mode is very similar but CFB mode is found to surprisingly perform almost 6.5 times slower than both CBC and OFB.

VII. Strength of AES

In the properly implemented versions of the AES algorithm, brute force is the only known practical attack to date. It requires the attacker to try every possible combination of encryption key until the key space is exhausted. On average, to brute-force attack AES-256, one would need to try all 2^{256} keys and even if only one second is required to try a single key, the total time required to perform this attack becomes astronomically huge (more than 10^{69} years). Different types of attacks have been proposed in the literature on reduced round versions of AES. One of the major attacks is side-channel attacks which work by gathering information leaks in the system while the algorithm is in operation. These attacks are also preventable if a proper implementation of the algorithm is used. Hence, most of the attacks are carried out on AES implementation instead of the algorithm itself [13].

A. Square Attacks

These types of attacks are possible only on the reduced round versions of AES. It is faster than an exhaustive key search for the AES versions of up to 6 rounds [5]. For this attack, a "multiset" of plaintexts is carefully chosen to have certain properties [14]. It can be applied to AES with a relatively small number of chosen plaintext-ciphertext pairs reduced to less than six rounds of encryption in the case of AES-128 [12]. The inner workings of this attack are out of the scope of this paper and one can find more about it in [5] and [12].

Tunstall et al. were able to use this attack by using 256 distinct plaintexts that are equal in fifteen bytes. Since, after computing two rounds of AES the described property will be valid between all possible pairs, the XOR sum of the 256 bytes at each index will therefore be equal to zero. They applied the square attack on AES-128, 192, and 256 on various reduced round versions and were able to halve the time complexity of the attack by using the "partial sum" method [12].

B. Interpolation Attacks

Interpolation attack is a type of cryptanalytic attack against block ciphers. The cipher is modeled using a high-order polynomial in this kind of attack [14]. The complicated expression of the S-box in GF(2^8), in combination with the effect of the diffusion layer, prohibits these types of attacks for more than a few rounds. AES is thus made to be secure by design against any kinds of interpolation attacks [5].

C. Related-key Attacks

In related-key attacks, the attacker encrypts each plaintext with two keys, and the relation between two keys is known but the keys themselves are unknown. The key schedule of AES, with its diffusion property and non-linearity, makes this type of attack very improbable however, if the algorithm behind AES is used for proposes other than encryption, like hashing, a successful related-key attack is able to break the hash function [5, 14].

VIII. Besides Encryption

This section discusses about some other functions the algorithm behind AES can be used for.

A. Message Authentication Code (MAC)

A MAC generates a cryptographically secure authentication tag of a given message. They are used for protecting the integrity of a message during transit. AES can be used as a MAC algorithm by using it as the block cipher in a CBC-MAC algorithm [5]. Minematsu et al. were able to demonstrate that MACs implemented using only 4-round versions of AES are provably secure and 1.4 to 2.5 times faster than the previous MAC modes of AES such as the CBC MAC-AES [15].

B. Hash Functions

A hash function is a one-way function that maps an arbitrary-length message to a fixed-length output. Hash functions are practically infeasible to invert as they are one-way functions and the original information is lost while hashing.

Daemen et al. have suggested that AES can be used as an iterated hash function by using it as the round function [5]. An overview of the AES as a hash function for confirming the identity of software resident on a computer system is done in [16]. However, according to Kaminsky et al., related-key attacks are likely when a block cipher is used as a part of cryptographic hash function [14]. Due to this vulnerability, they suggest not using AES for hash-function structure. Schläffer et al. have provided an analysis of the security of hash functions and they propose a new attack strategy called the rebound attack [17].

C. Pseudo-random Generator (PRNG)

PRNGs have been widely used in applications such as computer simulation, modeling, statistics and cryptography [19]. They are important in practice for their speed in number generation and their reproducibility. There exist many ways to use AES as a PRNG. Daemen et al. suggested one such implementation of AES using a block length and cipher key of 256 bits in [5]. The state is updated by applying AES using the Cipher Key and the first 128 bits of the state are output as a pseudorandom number. Salmon et al. have also proposed a counter based PRNG based on AES which is able to enjoy parallel scalability [18].

IX. Conclusion

The study of AES done in this work discussed the main steps of the algorithm behind AES and implemented its different modes. The resulting experiments demonstrate that decryption is faster than encryption and furthermore CFB mode was found to be slower than all other implemented modes. CTR mode couldn’t be implemented during this study but it is expected to be faster than all other modes since it is the only mode in which both encryption and decryption parts are parallelizable.

The strength of AES against today’s computing power is very robust and AES is immune to all of the cryptanalytic attacks in the literature so far. Most attacks were found to be attacking the implementation of the AES instead of the actual algorithm behind it. Finally, we discussed some other fields in which AES could be used like generating MAC, hash functions, and pseudorandom numbers. In order to match the scale and scope of this project, the implementation of AES from scratch was not possible. In the future, we plan to implement AES in JAVA or C and run the algorithm on more varying types of data files. CTR mode of AES could also be implemented parallelly in the future using an implementation found in [24]. We predict CTR to be the fastest mode of AES if implemented parallelly and no noticeable differences to be seen in encryption and decryption performance of CTR parallel. This work aims to provide a general introduction to AES and its performance levels on various modes.

References

[1] C. Paar and J. Pelzl. Understanding Cryptography : Textbook for Students and Practitioners. London, Springer, 2010

[2] Blazhevski, Dobre, et al. "Modes of operation of the AES algorithm." (2013): 212-216.

[3] https://www.giac.org/paper/gsec/388/mathematics-rijndael/101001

[4] http://www.crypto-it.net/eng/theory/modes-of-block-ciphers.html

[5] Daemen, Joan, and Vincent Rijmen. "AES proposal: Rijndael." (1999)

[6] Benvenuto, Christoforus Juan. "Galois field in cryptography." *University of Washington* 1.1 (2012): 1-11.

[7] https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#cfb-mode

[8] https://github.com/ricmoo/pyaes

[9] https://www.educative.io/edpresso/what-is-the-ofb

[10] http://styere.xyz/JS-AES.html

[11] Van Tilborg, Henk CA, and Sushil Jajodia, eds. *Encyclopedia of cryptography and security*. Springer Science & Business Media, 2014.

[12] Tunstall, Michael. "Improved" Partial Sums"-based Square Attack on AES." *SECRYPT* 2012 (2012): 25-34.

[13] https://www.cs.rit.edu/~spr/gdn2010/aesark.pdf

[14] Kaminsky, Alan, Michael Kurdziel, and Stanisław Radziszowski. "An overview of cryptanalysis research for the advanced encryption standard." *2010-MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*. IEEE, 2010.

[15] Minematsu, Kazuhiko, and Yukiyasu Tsunoo. "Provably secure MACs from differentially-uniform permutations and AES-based implementations." *International Workshop on Fast Software Encryption*. Springer, Berlin, Heidelberg, 2006.

[16] Hansen, Randy R., et al. *Implementation of the AES as a Hash Function for Confirming the Identity of Software on a Computer System*. No. PNNL-14170. Pacific Northwest National Lab., Richland, WA (US), 2003.

[17] Schläffer, Martin. *Cryptanalysis of AES-based hash functions*. na, 2011.

[18] Salmon, John K., et al. "Parallel random numbers: as easy as 1, 2, 3." *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 2011.

[19] S. Ulam, R. Richtmeyer, and J. von Neumann. Statistical methods in neutron diffusion. Technical Report LAMS-551, Los Alamos Scientific Laboratory, April 1947.

[20] http://www.distributed.net/images/d/d7/19990119_-_PR_-_release-des3.pdf

[21] Simmons, Gustavus J. *AES*. Encyclopædia Britannica, 31 July 2014, www.britannica.com/topic/AES.

[22] https://www.solarwindsmsp.com/blog/aes-256-encryption-algorithm

[23] https://competitions.cr.yp.to/aes.html

[24] Tran, Nhat-Phuong, et al. "Parallel execution of AES-CTR algorithm using extended block size." *2011 14th IEEE International Conference on Computational Science and Engineering*. IEEE, 2011.

[25] Kak, Avi. "Lecture Notes on "Computer and Network Security"." *Purdue University* (2015).

[26] https://www.cs.uaf.edu/2015/spring/cs463/lecture/03_23_AES.html