Advanced Encryption Standard Enhancement with Output Feedback Block Mode Operation

Renas R. Asaad, Saman M. Abdulrahman, and Ahmed A. Hani

Department of Computer Science, College of Computer & Information Technology, Nawroz University, Duhok, Iraq

ABSTRACT

There is a great research in the field of data security these days. Storing information digitally in the cloud and transferring it over the internet proposes risks of disclosure and unauthorized access; thus, users, organizations, and businesses are adapting new technology and methods to protect their data from breaches. In this paper, we introduce a method to provide higher security for data transferred over the internet, or information based in the cloud. The introduced method for the most part depends on the Advanced Encryption Standard (AES) algorithm, which is currently the standard for secret key encryption. A standardized version of the algorithm was used by The Federal Information Processing Standard 197 called Rijndael for the AES. The AES algorithm processes data through a combination of exclusive-OR operations (XOR), octet substitution with an S-box, row and column rotations, and MixColumn operations. The fact that the algorithm could be easily implemented and run on a regular computer in a reasonable amount of time made it highly favorable and successful. In this paper, the proposed method provides a new dimension of security to the AES algorithm by securing the key itself such that even when the key is disclosed; the text cannot be deciphered. This is done by enciphering the key using Output Feedback Block Mode Operation. This introduces a new level of security to the key in a way, in which deciphering the data requires prior knowledge of the key and the algorithm used to encipher the key for the purpose of deciphering the transferred text.

KEY WORDS: Advanced Encryption Standard, Output Feedback Block, Block Cipher Mode Operation, Cryptography.

1. INTRODUCTION

Cryptography is the science oriented with privacy and security. Which is made up of several cryptosystems, these cryptosystems are basically a collection of algorithms that aim at securing information and data. Recently, cryptosystems are wide utilized in all branches of digital technology, electronic mails, and internet banking. This paper shortly discusses the most favorable cryptosystems and investigates the most common private-key cipher. On January 2, 1997, the National Institute of Standards and Technology (NIST) held a challenge for a new encryption standard. The previously used standard, Data Encryption Standard (DES), was no longer capable of providing sufficient for security. The algorithm had been used since November 23, 1976. Since then, computers have developed providing greater computer power, thus the algorithm was rendered not safe. In 1998 a specially developed computer, the DES cracker was developed by the Electronic Frontier Foundation for approximately $ 250,000 and winning the RSA DES Challenge II-2 (Kaufman et al., 2002).

The alternatives for a new encryption standard were Triple DES and International Data Encryption Algorithm. However, these alternatives were slow and not free to implement due to patent rights. NIST required an algorithm that provided high security that is efficient, flexible, and easy to implement and free to use (Dar et al., 2014).

About 3 years into the contest, NIST chose the Rijndael algorithm (Dar et al., 2014) which is pronounced "Rhine Dahl" in English (National Institute of Standards and Technology, 2001).

Academic Journal of Nawroz University (AJNU)
Volume 6, No 3(2017), 10 pages
Received 1 November 2016; Accepted 14 December 2016
Regular research paper: Published 18 July 2017
Corresponding author's e-mail: renas.rekany@nawroz.edu.krd

On November 26, 2001, the Rijndael algorithm was announced as the new encryption standard by the Federal Information Processing Standards Publication 197. A collaboration of efforts from two Belgian cryptographers, Vincent Rijmen and Joan Daemen, resulted in an algorithm in which replaced the old Data Encryption Standard (DES). This standard was called Advanced Encryption Standard (AES) and is currently still the standard for encryption (Daemon and Rijman, 2003).

2. RELATED WORKS

These days, AES is used in a lot of research and development. Moreover, individual implementations of the AES exist. In this article, output feedback (OFB) is used to encrypt the plain text and the key producing ciphered text and ciphered key. The ciphered key and ciphered text are inserted into AES for encryption. The AES algorithm resumes as usual until the final cipher text is produced which is sent to the receiver along with the original key. Note that the described algorithm mainly focuses on algorithm privacy; meanwhile, the key is considered public. Hence, the algorithm emphases on improving the AES algorithm by encrypting the key and plain text using OFB; these results in a higher security algorithm based on OFB enhancement to AES security (Daemon and Rijman, 2003).

3. THE PROPOSED APPROACH OF ENCIPHERING

3.1. AES Algorithm

AES is a reversible algorithm, i.e., encryption and decryption steps can be performed in reverse order for each task. AES is easier to implement and explain because it operates on bytes. The key used in AES is processed through a process called key expansion, where each key is expanded into individual subkeys in each iteration of the algorithm, which is described later in the paper.

As pointed earlier, AES is an iterated block cipher. This indicates that the same operations are performed many times on a fixed number of bytes. These operations can be subdivided to the below functions:

ADD ROUND KEY
BYTE SUB
SHIFT ROW
MIXCOLUMN

An iteration of the above steps is called a round. The number of iteration used to encrypt some text is relatively dependent on the key size, as shown in Table 1.

At the last iteration MixColumn step is not performed, this is to make the algorithm reversible at decryption.

Encryption (Rijndael Block and Key)

The AES algorithm applies to a fixed block and key size; the block and key size can be one of the followings. Block sizes can be of 128, 168, 192, 224, and 256 bits, while the key sizes can be of 128, 192, and 256 bits (Dar et al., 2014). AES-128 is the standard encryption where a 128-bit block and key are used. The block size is usually referred to as Nb which is the number of columns in a block, while Nk refers to the key size. Each row in an Nb block column consists of four cells of 8 bytes each for AES-128 (DI Management Services Pty Limited, 2003).

In AES-128, each block consists of 128 bits. Nb can be determined by dividing 128 by 32, equals the number of bytes in each column. Hence, Nb is 4. The original plain text is stored in bytes in a block.

The Table 2 is a encryption cipher using a 16-byte key.

Decryption (Rijndael Block and Key)

Decryption process in AES is simple after understanding the encryption. Basically, the decryption is just the encryption reversed. The design of the algorithm allows the two processes to be invertible, hence applying the steps of encryption in the reverse order decrypts the cipher text. Thus, decryption starts at the last round of encryption with the last round key. When processing, each round does the process backward. Hence, the last key is added first to the last round. The addition inverse is addition itself, which is neat. Afterward the MixColumn step is applied. Consider that MixColumn is not applied at the last iteration. Furthermore, the MixColumn inverse table is used (DI Management Services Pty Limited, 2003). The MixColumn inverse table is generated using another matrix using a similar process the MixColumn table was generated. However, there are no shortcuts to generate the MixColumn inverse table. Thus, the matrix multiplication needs to be performed in the field GF 2^8.

The Table 3 is a AES decryption cipher using a 16-byte key.

3.2. AES Cipher Functions

Rijndael Rounds

Each byte of a block at a given iteration is XORed with the corresponding byte from the expanded key. This is

TABLE 1
Key Size Table

Key size (bytes)	Block size (bytes)	Rounds
16	16	10
24	16	12
32	16	14

TABLE 2
Rounds and Functions Encryption

Round	Function
-	Add Round Key(State)
0	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
1	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
2	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
3	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
4	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
5	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
6	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
7	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
8	Add Round Key(MixColumn(Shift Row(Byte Sub(State))))
9	Add Round Key(Shift Row(Byte Sub(State)))

done for the 16 bytes at a given round. The bytes from the expanded key are never reused. Hence, once the 16 bytes of the first block are XORed with the expanded key's 16 bytes then the bytes 1-16 from the expanded key are never used again (Trenholme, n.d). At the second iteration, the Add Round Key function is called on bytes 17-32 which are XORed against the state.

The first time Add Round Key gets executed.

State	1	2	3	4	5	6	7	8
	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR
Exp Key	1	2	3	4	5	6	7	8
State	9	10	11	12	13	14	15	16
	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR
Exp Key	9	10	11	12	13	14	15	16

The second time Add Round Key gets executed.

State	1	2	3	4	5	6	7	8
	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR
Exp Key	17	18	19	20	21	22	23	24
State	9	10	11	12	13	14	15	16
	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR
Exp Key	25	26	27	28	29	30	31	32

And so on for each round of execution.

TABLE 3
Rounds and Functions Decryption

Round	Function
-	Add Round Key(State)
0	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
1	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
2	Mix Column(Add Round Key(Byte Sub(Shift Row (State))))
3	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
4	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
5	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
6	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
7	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
8	MixColumn(Add Round Key(Byte Sub(Shift Row (State))))
9	Add Round Key(Byte Sub(Shift Row(State)))

During decryption, the process is reversed. The state (16 bytes of ciphertext) is first XORed against the last 16 bytes of the expanded key, then the second last 16 bytes and so on.

Byte Sub

During encryption, each value of the state is replaced with the corresponding SBOX value, as shown in Table 4 .

During decryption, each value in the state is replaced with the corresponding inverse of the SBOX, as shown in Table 5.

Rijndael Shift Row

Shift Row operation is basically arranging the state in a matrix and performing a circular shift for each row. The circular shift just moves each byte one space over. Thus, it is not a bitwise shift, i.e., a byte in position three will be moved to position four and so on for the rest. However, for the last byte, it is placed in the first position of the state (Trenholme, n.d).

In detail: The state is arranged in a 4×4 matrix (square). The process is a little misleading since the matrix is formed in a vertical order while it's shifted in a horizontal manner.

So, bytes 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Will form a matrix:

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Depending on a row at a given state, reach row is moved over one, two or three each row is then moved over (shifted) 1, 2 or 3 spaces over to the right, depending on the row of the state. First, row is never shifted

Row 1	0
Row 3	1
Row 3	2
Row 4	3

TABLE 4
AES S-Box Lookup

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

For example, HEX 19 would get replaced with HEX D4

TABLE 5
AES Inverse S-Box

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

For example, HEX D4 would get replaced with HEX 19

The following table shows how the individual bytes are first arranged in the table and then moved over (shifted).

Blocks 16 bytes long:

From				To			
1	5	9	13	1	5	9	13
2	6	10	14	6	10	14	2
3	7	11	15	11	15	3	7
4	8	12	16	16	4	8	12

During decryption, the same process is reversed and all rows are shifted to the left:

From				To			
1	5	9	13	1	5	9	13
2	6	10	14	14	2	6	10
3	7	11	15	11	12	3	7
4	8	12	16	8	12	16	4

Rijndael MixColumn

This may be the most sophisticated step in the process, which is difficult to explain and understand. It's made up of two parts, first involves multiplying specific parts of the state against specific parts of the matrix, while the second involves how the multiplication is carried out over what is called a Galois Field (Trenholme, n.d).

Matrix multiplication

As described in the Shift Row Function, the state is put into a four-row table. Each column of the table at a time is multiplied against every value of the matrix (a total of sixteen multiplications); afterward the results of the multiplications are XORed together to produce four result bytes for the later state, so four bytes of input, 16 multiplication, 12 XORs, and four bytes of output. That being said, each one matrix two is multiplied against each value of the state column at one time (Trenholme, n.d).

Multiplication matrix

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

16-byte state

b1	b5 b9 b13
b2	b6 b10 b14
b3	b7 b11 b15
b4	b8 b12 b16

Below is a demonstration of how the first column state bytes are multiplied against the matrix:

b1=(b1*2) XOR (b2*3) XOR (b3*1) XOR (b4*1)
b2=(b1*1) XOR (b2*2) XOR (b3*3) XOR (b4*1)
b3=(b1*1) XOR (b2*1) XOR (b3*2) XOR (b4*3)
b4=(b1*3) XOR (b2*1) XOR (b3*1) XOR (b4*2)

(b1=specifies the first byte of the state)

Below is a demonstration of how the second column state bytes are multiplied against the matrix:

b5=(b5*2) XOR (b6*3) XOR (b7*1) XOR (b8*1)
b6=(b5*1) XOR (b6*2) XOR (b7*3) XOR (b8*1)
b7=(b5*1) XOR (b6*1) XOR (b7*2) XOR (b8*3)
b8=(b5*3) XOR (b6*1) XOR (b7*1) XOR (b8*2)

And so on until all columns of the state are exhausted.

Galois Field Multiplication

The multiplication in the previous section is performed over a Galois Field. The mathematics used in the multiplication is beyond the scope of this article. Instead, this section will focus on the multiplication implementation

which is done easily using the two Tables 7 and 8 (Trenholme, n.d, Jain, 2011).

The multiplication result can be easily obtained from a lookup of the L table, by adding the results, and a look up at table E. The addition process is a simple mathematical addition and not a bitwise AND operation. Numbers being multiplied through the MixColumn function are first converted to HEX which must form a minimum of two-digit Hex number. The first digit of the Hex value is used for the vertical index, while the second is used on the horizontal index. If the value multiplied is converted into a single hex value, then a 0 value is added to the left of it to represent the 0-vertical index (Jain, 2011).

For example, consider the hex values multiplied are AF * 8; first, we perform a look up at table L for (AF) and (08) which returns (B7) and (4B), respectively, afterward the values are simply added together. If the resultant value is greater than (FF), we subtract (FF) from the addition result. For example, B7+4B=102 which is greater than FF, so the following is performed: 102-FF=03. Then, 03 is used to lookup the E table, as before the first digit represents the vertical index while the second represent the horizontal index. For example, (03)=0F (Shneier, 2009).

Therefore, the result of multiplying AF * 8 over a Galois Field is 0F.

Two last exceptions are that:

Any number multiplied by one is equal to its self and does not need to go through the above procedure.

For example: FF*1=FF

Any number multiplied by zero equals zero.

Rijndael MixColumn Inverse

During decryption, the MixColumn the multiplication matrix is changed from inverse matrix (Table 6) to:

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

TABLE 6
Fixed Matrix

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

TABLE 7
E Table

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

TABLE 8
L Table

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

Other than the change to the matrix table the function performs the same steps as during encryption.

MixColumn Example

The following examples are denoted in HEX.

MixColumn example during encryption
Input=D4 BF 5D 30
Output (0)=(D4*2) XOR (BF*3) XOR (5D*1) XOR (30*1)
=E(L(D4)+L(02)) XOR E(L(BF)+L(03)) XOR 5D XOR 30
=E(41+19) XOR E(9D+01) XOR 5D XOR 30
=E(5A) XOR E(9E) XOR 5D XOR 30
=B3 XOR DA XOR 5D XOR 30**=04**
Output (1)=(D4*1) XOR (BF*2) XOR (5D*3) XOR (30*1)
=D4 XOR E(L(BF)+L(02)) XOR E(L(5D)+L(03)) XOR 30
=D4 XOR E(9D+19) XOR E(88+01) XOR 30
=D4 XOR E(B6) XOR E(89) XOR 30
= D4 XOR 65 XOR E7 XOR 30**=66**
Output (2)=(D4*1) XOR (BF*1) XOR (5D*2) XOR (30*3)
=D4 XOR BF XOR E(L(5D)+L(02)) XOR E(L(30)+L(03))
=D4 XOR BF XOR E(88+19) XOR E(65+01)
=D4 XOR BF XOR E(A1) XOR E(66)
=D4 XOR BF XOR BA XOR 50**=81**
Output (3)=(D4*3) XOR (BF*1) XOR (5D*1) XOR (30*2)
=E(L(D4)+L(3)) XOR BF XOR 5D XOR E(L(30)+L(02))
=E(41+01) XOR BF XOR 5D XOR E(65+19)
=E(42) XOR BF XOR 5D XOR E(7E)
=67 XOR BF XOR 5D XOR 60**=E5**

MixColumn during decryption
Input 04 66 81 E5
Output (0)=(04*0E) XOR (66*0B) XOR (81*0D) XOR (E5*09)
=nE(L(04)+L(0E)) XOR E(L(66)+L(0B)) XOR E(L(81)+L(0D)) XOR E(L(E5)+L(09))
=E(32+DF) XOR E(1E+68) XOR E(58+EE) XOR E(20+C7)
=E(111-FF) XOR E(86) XOR E(146-FF) XOR E(E7)
=E(12) XOR E(86) XOR E(47) XOR E(E7)
=38 XOR B7 XOR D7 XOR 8C**=D4**
Output (1)=(04*09) XOR (66*0E) XOR (81*0B) XOR (E5*0D)
=E(L(04)+L(09)) XOR E(L(66)+L(0E)) XOR E(L(81)+L(0B)) XOR E(L(E5)+L(0D))
=E(32+C7) XOR E(1E+DF) XOR E(58+68) XOR E(20+EE)
=E(F9) XOR E(FD) XOR E(C0) XOR E(10E-FF)
=E(F9) XOR E(FD) XOR E(C0) XOR E(0F)
=24 XOR 52 XOR FC XOR 35**=BF**
Output (2)=(04*0D) XOR (66*09) XOR (81*0E) XOR (E5*0B)
=E(L(04)+L(0D)) XOR E(L(66)+L(09) XOR E(L(81)+L(0E)) XOR E(L(E5)+(0B))
=E(32+EE) XOR E(1E+C7) XOR E(58+DF) XOR E(20+68)
=E(120-FF) XOR E(E5) XOR E(137-FF) XOR E(88)
=E(21) XOR E(E5) XOR E(38) XOR E(88)
=34 XOR 7B XOR 4F XOR 5D**=5D**
Output (3)=(04*0B) XOR (66*0D) XOR (81*09) XOR (E5*0E)
=E(L(04)+L(0B)) XOR E(L(66)+L(0D)) XOR E(L(81)+L(09)) XOR E(L(E5)+L(0E))
=E(32+68) XOR E(1E+EE) XOR E(58+C7) XOR E(20+DF)
=E(9A) XOR E(10C-FF) XOR E(11F-FF) XOR E(FF)
=E(9A) XOR E(0D) XOR E(20) XOR E(FF)
=2C XOR F8 XOR E5 XOR 01=**30**

3.3. Rijndael Key Expansion

Before any of the encryption or decryption processes are carried out; the key used must go through an expansion process. The resulted key from the expansion is used in the Add Round key function explained above. At each Add Round function call, a different part of the expanded key is XORed with the current state (Shneier, 2009). Thus, the Expanded Key must be large enough in order for this to work; it must provide enough key material for each time the Add Round Key function is called, which is called at each iteration plus one extra call at the beginning of the algorithm. Hence, the expanded key size must always be (16*(number of round+1)). The 16 in this function equals the size of the block in bytes, which provides sufficient key material for each round plus the one at the beginning (Daemen and Rijmen, 2013).

In this algorithm, the actual key is stretched out to provide enough key space for the algorithm because the actual key is smaller than the subkeys used at each iteration.

The key expansion routine executes a maximum of 4 consecutive functions. These functions are:

ROT WORD
SUB WORD
RCON
EK
K

An iteration of the above steps is called a round. The number of rounds of the key expansion algorithm depends on the key size, as shown in Table 9.

The first bytes of the expanded key and the actual key are identical. Consider that the key is 16 bytes longs, after the key is expanded; the first 16 bytes of the expanded key are the same as the original key. At each round, 4 bytes are added to the expanded key except the first rounds. The 4 bytes added at each round are taken as input in at the next round and 4 other bytes are returned. It's important to note that all 4 functions are always called in each round:

TABLE 9
Key Expansion

Key size (bytes)	Block size (bytes)	Expansion algorithm rounds	Expanded bytes/round	Rounds Key Copy	Rounds Key Expansion	Expanded Key (bytes)
16	16	44	4	4	40	176
24	16	52	4	6	46	208
32	16	60	4	8	52	240

4 Rounds for a 16 byte Key
6 Rounds for a 24 byte Key
8 Rounds for a 32 byte Key

At the remaining rounds, the k function result is XORed with the EK function result. Except when the key is 32 bytes long, the Subword function is called additionally every 8 rounds beginning at round 13.

3.4. Rijndael Key Expansion Functions

Rot Word (4 bytes)

This does a circular shift on 4 bytes similar to the Shift Row Function.

1, 2, 3, 4 to 2, 3, 4, 1

Sub Word (4 bytes)

This step applies the S-box value substitution as described in bytes sub function to each of the 4 bytes in the argument.

Rcon((Round/(KeySize/4))-1)

This function returns a 4 byte value based on the following table

Rcon(0)=01000000
Rcon(1)=02000000
Rcon(2)=04000000
Rcon(3)=08000000
Rcon(4)=10000000
Rcon(5)=20000000
Rcon(6)=40000000
Rcon(7)=80000000
Rcon(8)=1B000000
Rcon(9)=36000000
Rcon(10)=6C000000
Rcon(11)=D8000000
Rcon(12)=AB000000
Rcon(13)=4D000000
Rcon(14)=9A000000

For example, for a 16-byte key Rcon is first called in the 4th round

$$(\mathbf{4}/(\mathbf{16}/4))\text{-}1=0$$

In this case, Rcon will return 01000000

For a 24 byte, key Rcon is first called in the 6th round

$$(\mathbf{6}/(\mathbf{24}/4))\text{-}1=0$$

In this case, Rcon will also return 01000000.

EK(Offset)

EK function returns 4 bytes of the Expanded Key after the specified offset. For example, if offset is 0 then EK will return bytes 0,1,2,3 of the expanded key.

K(Offset)

K function returns 4 bytes of the Key after the specified offset. For example, if offset is 0 then **K** will return bytes 0,1,2,3 of the expanded key.

3.5. Rijndael Key Expansion Algorithm

The Table 10 is a description of the key expansion in a table formation. This is because the algorithm depends on the key size, thus it's challenging to explain in writing (Daemen and Rijmen, 2013; Wagner, 2002).

Notice that most values that change in the tables below equal the current round number. This makes implementation in code much easier as these numbers can easily be replaced with loop variables.

16-byte Key Expansion

Each round (except rounds 0, 1, 2 and 3) will take the result of the previous round and produce a 4-byte result for the current round. Notice the first 4 rounds simply copy the total of 16 bytes of the key, as shown in Table 11.

4. OFB

OFB Block Mode generates a synchronous stream cipher from a block cipher. It produces key stream bocks, which are XORed with blocks of plain text to make cipher text. Like any stream cipher algorithm, a bit flipped in ciphertext results in a bit flipped in the same location in the plain text. This allows most error correcting codes to function normally even when encryption is not applied yet (Daemen and Rijmen, 2013; McGill, 2000; Wagner, 2002).

Because of the symmetry of the XOR operation, encryption and decryption are exactly the same:

$C_J=P_J \oplus O_J$
$P_J=C_J \oplus O_J$
$O_J=E_K(I_J)$
$I_J=O_{J-1},\ldots\ldots I_0=IV$

Example for 8-bits input when the initial vector=01001101 and initial P_0= 11100101:

IV=01001101
P_0=11100101
C_0=IV=01001101
$C_1=E_K\ (P_0 \oplus C_{i-1})$
C_1= 01001101 ⊕ 11100101=10101000

TABLE 10
Key Expansion Formula

Field	Description
Round	A counter representing the current step in the key expansion algorithm, think of this as a loop counter
Expanded Key Bytes	Expanded key bytes effected by the result of the function(s)
Function	The function(s) that will return the 4 bytes written to the effected expanded key bytes

TABLE 11
Functions and Key Expansion

Round	Expanded Key Bytes				Function
0	0	1	2	3	K(0)
1	4	5	6	7	K(4)
2	8	9	10	11	K(8)
3	12	13	14	15	K(12)
4	16	17	18	19	Sub Word (Rot Word (EK((4-1)*4))) XOR Rcon((4/4)-1) XOR EK((4-4)*4)
5	20	21	22	23	EK((5-1)*4) XOR EK((5-4)*4)
6	24	25	26	27	EK((6-1)*4) XOR EK((6-4)*4)
7	28	29	30	31	EK((7-1)*4) XOR EK((7-4)*4)
8	32	33	34	35	Sub Word (Rot Word (EK((8-4)*4))) XOR Rcon((8/4)-1) XOR EK((8-4)*4)
9	36	37	38	39	EK((8-1)*4) XOR EK((9-4)*4)
10	40	41	42	43	EK((10-1)*4) XOR EK((10-4)*4)
11	44	45	46	47	EK((11-1)*4) XOR EK((11-4)*4)
12	48	49	50	51	Sub Word (Rot Word (EK((12-4)*4))) XOR Rcon((12/4)-1) XOR EK((12-4)*4)
13	52	53	54	55	EK((13-1)*4) XOR EK((13-4)*4)
14	56	57	58	59	EK((14-1)*4) XOR EK((14-4)*4)
15	60	61	62	63	EK((15-1)*4) XOR EK((15-4)*4)
16	64	65	66	67	Sub Word (Rot Word (EK((16-4)*4))) XOR Rcon((16/4)-1) XOR EK((16-4)*4)
17	68	69	70	71	EK((17-1)*4) XOR EK((17-4)*4)
18	72	73	74	75	EK((18-1)*4) XOR EK((18-4)*4)
19	76	77	78	79	EK((19-1)*4) XOR EK((19-4)*4)
20	80	81	82	83	Sub Word (Rot Word (EK((20-4)*4))) XOR Rcon((20/4)-1) XOR EK((20-4)*4)
21	84	85	86	87	EK((21-1)*4) XOR EK((21-4)*4)
22	88	89	90	91	EK((22-1)*4) XOR EK((22-4)*4)
23	92	93	94	95	EK((23-1)*4) XOR EK((23-4)*4)
24	96	97	98	99	Sub Word (Rot Word (EK((24-4)*4))) XOR Rcon((24/4)-1) XOR EK((24-4)*4)
25	100	101	102	103	EK((25-1)*4) XOR EK((25-4)*4)
26	104	105	106	107	EK((26-1)*4) XOR EK((26-4)*4)
27	108	109	110	111	EK((27-1)*4) XOR EK((27-4)*4)
28	112	113	114	115	Sub Word (Rot Word (EK((28-4)*4))) XOR Rcon((28/4)-1) XOR EK((28-4)*4)
29	116	117	118	119	EK((29-1)*4) XOR EK((29-4)*4)
30	120	121	122	123	EK((30-1)*4) XOR EK((30-4)*4)
31	124	125	126	127	EK((31-1)*4) XOR EK((31-4)*4)
32	128	129	130	131	Sub Word (Rot Word (EK((32-4)*4))) XOR Rcon((32/4)-1) XOR EK((32-4)*4)
33	132	133	134	135	EK((33-1)*4) XOR EK((33-4)*4)
34	136	137	138	139	EK((34-1)*4) XOR EK((34-4)*4)
35	140	141	142	143	EK((35-1)*4) XOR EK((35-4)*4)
36	144	145	146	147	Sub Word (Rot Word (EK((36-4)*4))) XOR Rcon((36/4)-1) XOR EK((36-4)*4)
37	148	149	150	151	EK((37-1)*4) XOR EK((37-4)*4)
38	152	153	154	155	EK((38-1)*4) XOR EK((38-4)*4)
39	156	157	158	159	EK((39-1)*4) XOR EK((39-4)*4)
40	160	161	162	163	Sub Word (Rot Word (EK((40-4)*4))) XOR Rcon((40/4)-1) XOR EK((40-4)*4)
41	164	165	166	167	EK((41-1)*4) XOR EK((41-4)*4)
42	168	169	170	171	EK((42-1)*4) XOR EK((42-4)*4)
43	172	173	174	175	EK((43-1)*4) XOR EK((43-4)*4)

$P_i=E_K (C_{i-1})\oplus C_i$
P_0= 01001101⊕ 10101000=11100101 to original P.

Each cipher operation in OFB depends on all the previous cipher operations executed as shown in Figure 1 and Figure 2, thus operations cannot be executed in parallel. However, the block cipher operations can be performed beforehand enabling the last step of the procedure to be executed in parallel whenever the plain text or cipher text is available. This is because the plain text or cipher text is only required for the final XOR operation (McGill, 2000).

It is possible to use Chaining Block Cipher Mode Operation, with a constant string of zeroes as input, to acquire a key stream of OFB mode. This allows exploiting the fast hardware implementations of CBC mode operations for OFB mode operations, which is highly advantageous to OFB algorithm. The average cycle length of OFB mode can be reduces by a factor of 2^{32} or more is OFB is used with a partial block feedback similar to Cipher Feedback Block Mode Operation. A mathematical model, originally proposed by Davies and Parkin, which was substantiated by experimental results a cycle length close

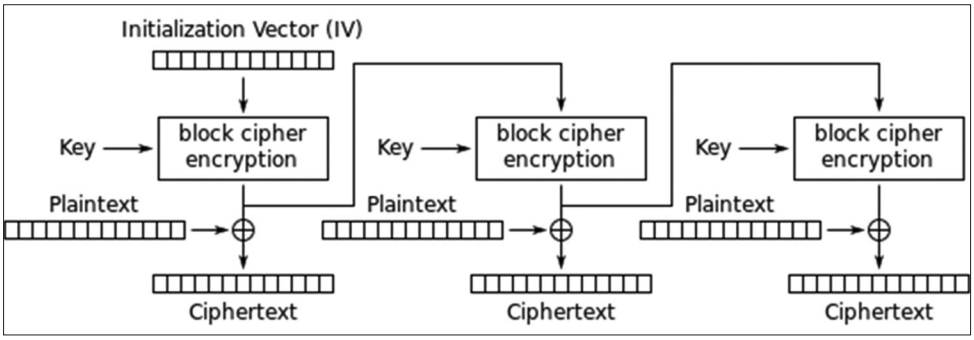

Fig. 1. Output Feedback Block Encryption.

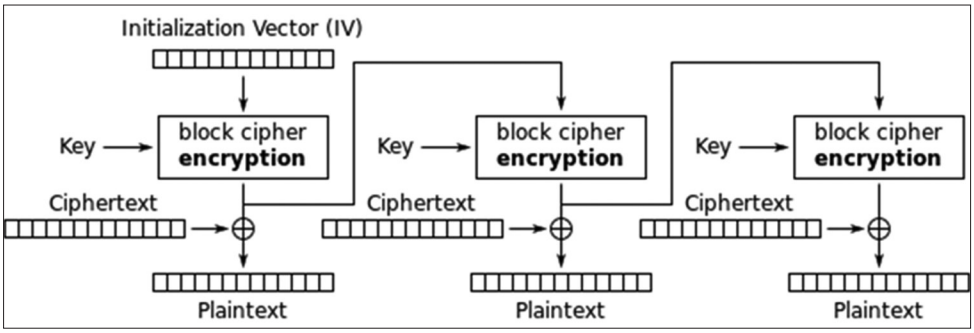

Fig. 2. Output Feedback Block Decryption.

to the obtainable maximum is achieved only by using a full feedback. Thus, truncated feedback support was removed from OFB specifications (Wagner, 2002).

5. EXPERIMENTAL RESULTS

AES has been identified as the new security standard after DES since it is very secure, easy to implement and understand. The security of AES relies on the different phases of encryption the algorithm provides on plain text to provide cipher text. The phases detailed above are as follows:

1. Add Round Key
2. Byte Sub
3. Shift Raw
4. MixColumn.

The result of these encryption steps is a very high-security cipher texts that are unbreakable by current computing power. However, like any ciphering algorithm, the security of the algorithm completely relies on the key security, if the key is disclosed, then the plain text can be easily obtained.

In this paper, the key vulnerability issue has been addressed by ciphering the key itself before use for the algorithm. This is done using OFB algorithm. The new hybrid algorithm provides additional security to AES by enciphering the plaintext using OFB first; in addition, it provides security for the key itself. Thus, the key can be considered public in this case, as if the key is disclosed, the key itself cannot decipher the ciphertext since it was originally ciphered used OFB before use for AES. That being said, the algorithm security in this case relies on the security of the OFB algorithm and its confidentiality.

6. CONCLUSION AND FUTURE WORK

In conclusion, data security is vital for such a technological era. Protecting digital data transferred and stored in a cloud-based system is imperative. The data can be related to a specific user or very sensitive governmental information. Thus, providing better security is crucial.

The custom algorithm proposed in this article provides additional security to the AES. Combining AES with OFB produces ciphers of great security that can be used for data transmission nowadays. It addresses the key vulnerability available is AES such that knowing the key is not enough to break the cipher. That being said, the security of this hybrid algorithm relies on key and algorithm confidentiality.

With regard to future work, the algorithm may be further enhanced by expanding the key size to 192 bits and AES Block size to 256 bits. This provides more complexity as the cipher becomes genuinely difficult to break.

REFERENCES

Dar, M.H., Mittal, P & Kumar, V. (2014). A comparetive study of cryptographic algorithms. *International Journal of Computer Science and Network,* 3(3), 1190.

Daemon, J & Rijman, V. (2003). Computer Security Resource Center. Available from: http://www.csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf. [Last accessed on 2016 Aug 08].

Daemen, J & Rijmen, V. (2013). *The Design of Rijndael: AES - The Advanced Encryption Standard.* Brussel: Springer Science and Business Media.

DI Management Services Pty Limited. (2003). DI Management. Available from: http://www.di-mgt.com.au/cryptopad.html. [Last accessed on 2016 Aug 08].

Jain, R. (2011). Washington University. Available from: http://www.cse.wustl.edu/~jain/cse571-11/ftp/l_05aes.pdf. [Last accessed on 2016 Aug 08].

Kaufman, C., Perlman, R & Speciner, M. (2002). *Network Security: Private Communication in a Public World.* Upper Saddle River, NJ: Prentice Hall Press.

McGill. (2000). RIJNDAEL Advanced Encryption Standard. Available from: http://www.cs.mcgill.ca/~kaleigh/computers/crypto_rijndael.html. [Last accessed on 2017 Apr 24].

National Institute of Standards and Technology. (2001). Computer Security Resource Center. Available from: http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf. [Last accessed on 2016 Aug 08].

Shneier, B. (2009). Shneier on Security. Available from: https://www.schneier.com/blog/archives/2009/07/another_new_aes.html. [Last accessed on 2016 Aug 08].

Trenholme, S. (n.d). Sam Trenholme. Available from: http://www.samiam.org/s-box.html. [Last accessed on 2016 Aug 08].

Trenholme, S. (n.d). Sam Trenholme. Available from: http://www.samiam.org/mix-column.html. [Last accessed on 2016 Aug 08].

Wagner, N.R. (2002). The University of Texas at San Antonio. Available from: http://www.cs.utsa.edu/~wagner/laws. [Last retrieved on 2016 Aug 08].