Implementation of 128-bit Advanced Encryption Standard algorithm using Verilog

Aditya Gupta, Akash Jha, Apoorva Verma, Apurba Prasad Padhy, Tushar Sahu

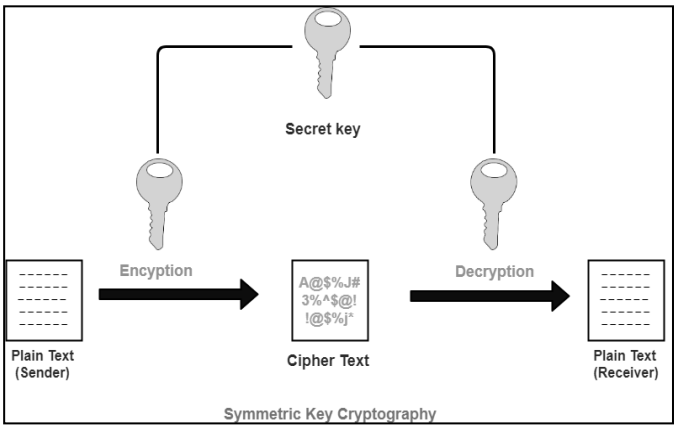
B. Tech I Year, Department of Electronics and Communication Engineering
Indian Institute of Technology, Roorkee

***Abstract*—Maintaining confidentiality and ensuring security of sensitive data is an open problem in today's age of information technology. It is also necessary to make sure that the transmitted data is not modified, intentionally or unintentionally, and that the authenticity of the sender is verified correctly. In this context, various cryptographic specifications have been developed to protect data from being accessed by anyone other than the sender and the intended receiver. In this project, we aim to implement 128-bit AES (Advanced Encryption Standard) algorithm using Verilog (only decryption).**

***Index Terms*— Advanced Encryption Standard, Decryption, Encryption, Security**

I. Introduction

Cryptography refers to the study of making communication between a sender and its intended receiver secure. The Data Encryption Standard (DES) was the first encryption algorithm to be adopted as a federal standard and was in use for nearly twenty-five years. However, there were several vulnerabilities with DES that with time came to the forefront. The hardware implementation of DES was not very efficient. While the theoretical feasibility of cracking DES had always been discussed, in 1997 a group of computer scientists cracked it at a challenge for the first time. The standard was susceptible to brute-force attacks and could be cracked in under two days by 1998. There were improvements made to the DES algorithm by virtue of using three times as many rounds (called 3DES). While 3DES was resistant to brute-force as well as cryptanalytic attacks, it was extremely slow. Moreover, both DES and 3DES used 64-bit blocks, which lead to compromises over efficiency and security. These drawbacks made 3DES unsuitable for long-term use, and an appropriate replacement was to be found for it



Symmetric Key Algorithm

In December 2001, the U.S. National Institute of Standards and Technology (NIST) announced AES (also known by its original name Rijndael, named after its developers) as the successor to DES, after two rounds of evaluation of around 15 submissions. The AES algorithm is a symmetric-key algorithm (also known as a private-key algorithm) which means that a common private key is used for encrypting the message at the sender's end, and for decrypting the same at the receiver's end. Such algorithms are known to be much faster than asymmetric-key algorithms, which require more computation. It is a subset of block ciphers, which means it converts a fixed-length block of plaintext data to ciphertext of the same length.

The AES algorithm was subjected to intense scrutiny over a long period of time before being accepted as the best alternative to replace DES. The following criteria were used to evaluate the capabilities of the AES algorithm:

- **Security**: This included both the actual security of the algorithm as well as that relative to other candidates. It also investigated any mathematical weaknesses in the algorithm and susceptibility to known methods of attacks.
- **Cost**: This referred to computational efficiency and memory requirements, as the algorithm should be flexible enough to work on less powerful hardware while not compromising on security.
- **Implementation Characteristics**: This referred to flexibility in terms of additional key sizes, block sizes, variety of processors on which the algorithm can be used, complexity etc.

II. Design procedure

In this project, we have implemented the decryption process for the AES algorithm. That is, we generate a 128-bit plaintext after taking in a 128-bit ciphertext and its associated 128-bit key as input. These blocks are represented in the form of 4×4 matrices, with each element representing one byte of data. The ordering of bytes within a matrix is column wise, that is the first four bytes of data occupy the first column and form the first word, and so on. For a 128-bit key as we have taken in this implementation, the cipher consists of 10 rounds, with the first 9 rounds consisting of four distinct transformations (Inv ShiftRows, Inv SubBytes, AddRoundKey, Inv MixColumns) and the final round consisting of three transformations (exclude Inv MixColumns from the final step). These rounds are preceded by an AddRoundKey stage (also referred to as Round

0). Before beginning the decryption process, the key is passed through a key expansion function, which produces 11 round keys, which are used as one of the inputs at every 'AddRoundKey' stage. The entire design can be summarized using a flowchart (shown in figure 1).

An important point to note is that while the decryption algorithm makes use of the expanded key in reverse order, the decryption process is not identical to the encryption process. This is a result of the structure of the AES algorithm.

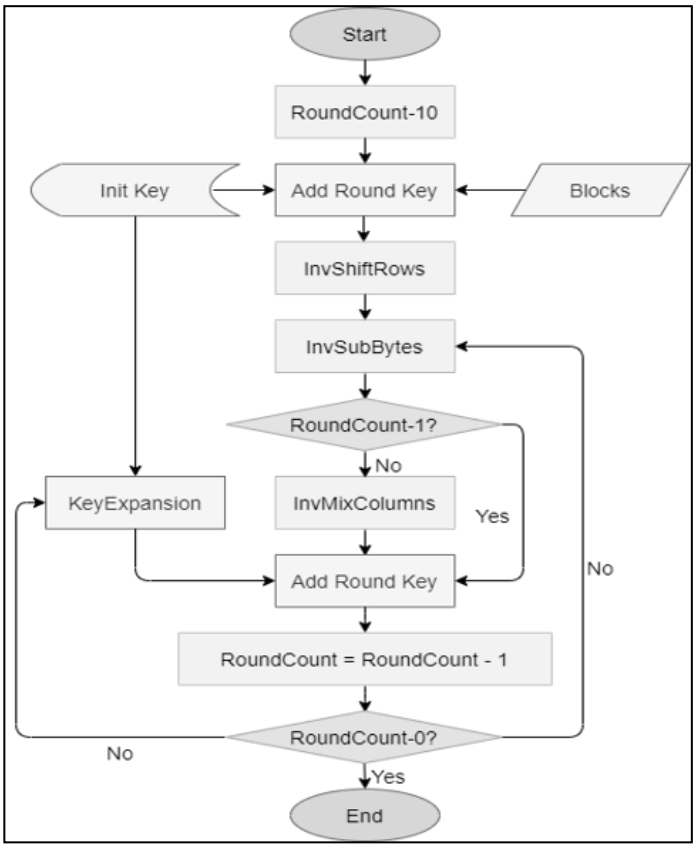


Figure 1: Flowchart of the AES Decryption process

The description of the inverse transformation functions involved in the decryption process along with the reasoning behind the corresponding cipher function is given as follows:

- **InvShiftRows:** The inverse shift row transformation performs circular shifts in such a way that the i^{th} row undergoes (i-1) 1-byte right shifts. The rationale behind the ShiftRows operation was to spread out 4 bytes of one column to different columns and introduce a degree of randomness in the cipher.

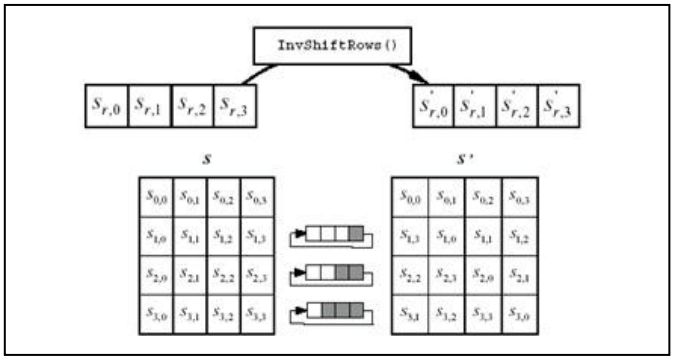


Figure 2: Inverse ShiftRow Operation

- **InvSubBytes:** The Inverse Substitute Byte transformation uses a 16×16 table (called InvS-Box) to find one-to-one replacement bytes for all possible bytes in the input state array. The table is derived from the inverse function over GF (2^8) and stays the same for all the rounds of decryption.

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

Figure 3: Inv S-Box Lookup Table

For calculating respective InvS-Box substitute of any input byte xy_{16}, say $(b_7b_6b_5b_4b_3b_2b_1b_0)_2$, from the ciphertext, a byte word mn_{16}, say $(b'_7b'_6b'_5b'_4b'_3b'_2b'_1b'_0)_2$, is calculated using the matrix equation shown in figure 4.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figure 4: Calculating InvS-Box Substitutes

The above matrix multiplication should be handled carefully. Unlike ordinary matrix multiplication, in the above equation, the products of elements of one row and one column of matrices to be multiplied are bitwise XORed to get each element of the subsequent product matrix, and the final addition is a bitwise XOR too. The InvS-Box substitute $x'y'_{16}$ is then calculated by taking the inverse of mn_{16} over GF (2^8). $x'y'_{16} = \{mn_{16}\}^{-1}$. The motive of this step during encryption rounds is to reduce the correlation between input and output bits at the byte level.

- **InvMixColumns:** The Inverse Mix Column transformation causes every byte in a column to affect every other byte in the state matrix. The transformation performs a linear operation on the columns of the state matrix. The state matrix is represented as column polynomials over GF (2^8) and the transformation consists of matrix multiplication of the state with a polynomial over a finite field. The inverse mix column transformation step is the only place in Rijndael's round transformation where the columns are mixed. This works after the AddRoundKey stage to ensure that all parts of the block affect each other. For a data block of 128 bits, the state matrix has 4 rows. Therefore, the columns of the state matrix are each viewed as the polynomial of degree 8 over GF (2^8) and multiplied modulo (x^4 +1).

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Figure 5: Inverse MixColumn Operation

- **Key Expansion:** The 128-bit AES decryption takes 128-bit (4 words) key as input and requires its expansion to 44 words for pre-round transformation and subsequent 10 rounds. In the key expansion process, the initial key is used to obtain next key, which is then used to get its next key. This is continued until we get total 11 keys. To get the next key from the previous one, the last column of the previous key (which is represented in the form of a 4×4 matrix), is passed as argument to another function. This function accepts 1 word (4 bytes) and copies it into a 1×4 matrix. A left shift operation and a substitute byte transformation is then performed on the obtained matrix. is then performed on the matrix. The matrix is then bitwise XORed with the round constant. The round constant has a size of 32 bits with first 8 bits different for each round and the next 24 bits being 0. Thus, round constant is different for different rounds. The matrix thus obtained is the output of the function. The output of the function is then XORed with the first column (first word) of the previous key to get the first column (first word) of the next key. The obtained word is then XORed with the second column (second word) of the previous key to get the second column (second word) of the next key, and so on. Thus, we get the next key (having 4 words) from the previous key. This process is repeated 10 times to get next 10 keys from the initial key.

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

Table 1: Round Constant Values used in Key Expansion

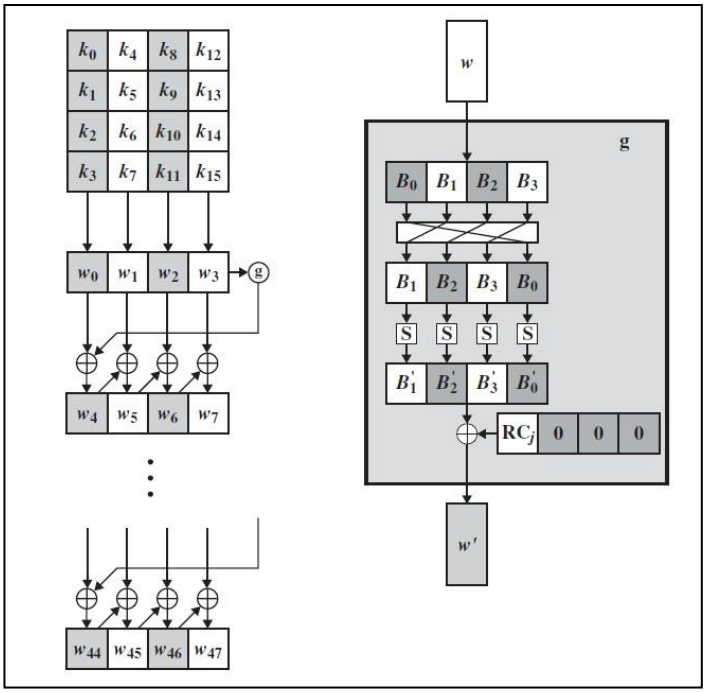


Figure 6: Key Expansion Operation
(note that for a 128-bit key, this process stops after obtaining 44 words)

- **AddRoundKey:** In the AddRoundKey transformation, the key is bitwise XORed with the state matrix. The 128-bit AES takes a 128-bit key as input which is the initial key. The key is expanded to total 11 such keys (each 128 bits) using the Key Expansion algorithm. These keys are then used in pre round transformation and subsequent 10 rounds. In AES decryption process, the last key (of the expanded keys) is used in the pre round transformation. The second last key is used in the first round and so on. Finally, the initial key is used in the last round. This is done to exactly reverse the encryption process.

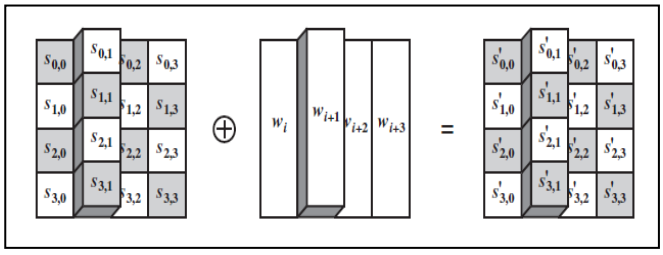


Figure 7: AddRoundKey Operation

III. Design Challenges and Trade-offs

While the Rijndael algorithm performs better than other encryption standards it was competing with as a replacement for DES, there are still some limitations which this algorithm suffers from. These limitations are more to do with the design

optimization of the algorithm, rather than some mathematical weakness which can be exploited to gain unauthorized access of the encrypted data. Most of it is to do with the inverse functions used in the decryption process.

1. From a software perspective, we note that the cipher and its inverse make use of different pieces of code and tables. This is observed with stages such as S-Box and Inv S-Box (different lookup tables), MixColumns and Inv MixColumns (different matrix transformations).
2. On a similar note, the hardware implementation of the inverse cipher can only partially reuse the circuitry from the cipher.
3. The developers of AES noted that the inverse cipher takes more code and cycles to be implemented on a smart card (like credit cards etc.), even if it is much faster than other ciphers.

IV. Simulation/Measured results

We have used Xilinx Vivado v2020.2 (64-bit) for running our simulations. We have taken the plaintext-ciphertext pairs from [1] for validation of our results. We have also used an online tool to generate additional pairs [4]. Table 1 denotes the plaintext-ciphertext combinations we must obtain to validate that our algorithm is working correctly.

From figures 8, 9 and 10, we observe that we have obtained the desired plaintext having given our ciphertext as well as key in the form of inputs.
(Note that *Test_State* refers to our ciphertext, *Test_Key* to our key and *Test_Result* to the generated plaintext)

V. Conclusions

We have noted how the Advanced Encryption Standard (AES) algorithm makes vast improvements over the DES algorithm as the current security standard, owing to its longer key size and a greater invulnerability to cryptanalytical attacks. Depending on the key length, we can have varying number of rounds for AES, and we see how each step adds an increasing degree of complexity to ensure security. In this project, we have also observed how this algorithm can be implemented on Verilog (and by its virtue, on hardware) successfully. Clearly, with no real mathematical weaknesses discovered yet, we can rely on the AES algorithm for keeping our data secure with a reasonable degree of confidence.

References

[1] William Stallings, 'Advanced Encryption Standard', Cryptography and Network Security: Principals and Practices, Fifth Edition, Pearson Education, Inc., 2011

[2] Joan Daemen and Vincent Rijmen, AES submission document on Rijndael, Version 2, September 1999.

[3] Samir Palnitkar, 'Verilog HDL: A Guide to Digital Design and Synthesis', SunSoft Press, 1996.

[4] AES Symmetric Cipher Online

S.No.	Ciphertext	Key	Expected Plaintext
1	29c3505F571420f6402299b31a02d73a	5468617473206d79204b756e67204675	54776f204f665204e696e652054776f
2	ff0b844a0853bf7c6934ab4364148fb9	0f1571c947d9e8590cb7add6af7f6798	0123456789abcedffedcba9876543210
3	ae7614a563d5f29824182476369ffff	f1571c9546ae74863ef9752467983344	63881d081750d77786fc1c42c0717c93

Table 2: Ciphertext-Key-Plaintext triplets that we should obtain after decryption

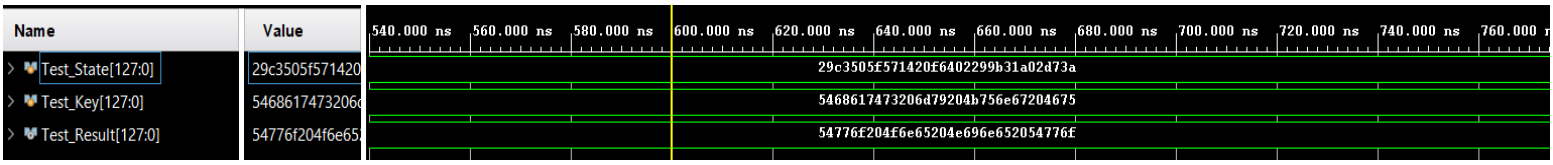


Figure 8: Simulation 1

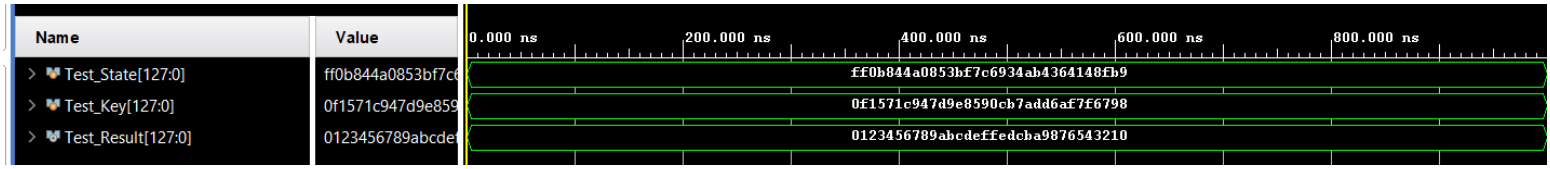


Figure 9: Simulation 2

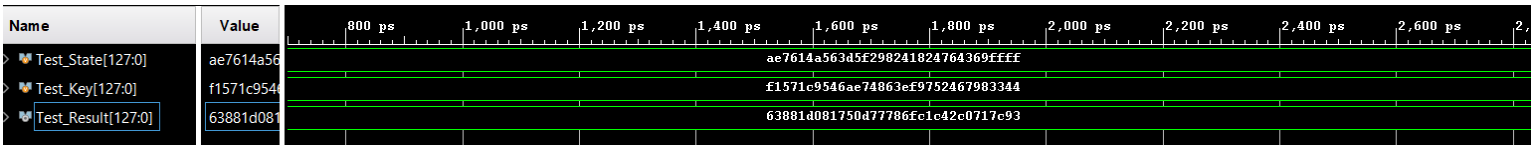


Figure 10: Simulation 3