

**PROJECT AIM**

**Design Encryption and Decryption scheme for modes of operation on Advanced Encryption Standard (AES-128) in Verilog**

**EXPLORATORY PROJECT**

**REPORT**

***Submitted By***

**Pranav Mittal (19095076)**

*Under the Guidance of*

**DR. NS RAJPUT SIR**

**(Department of Electronics Engineering)**

**DEPT. OF ELECTRONICS ENGINEERING,**

**IIT(BHU) VARANASI**

**CERTIFICATE**

This is to certify that this project report **“Design Encryption and Decryption scheme for modes of operation on Advanced Encryption Standard (AES-128) in Verilog”** is submitted by **Pranav Mittal** who carried out the project work under the supervision of **Rd. NS Rajput.** We approve this project for submission of the Exploratory Project, IIT(BHU) Varanasi.

**ACKNOWLEDGEMENT**

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide **Dr. NS Rajput sir**, Dept. of Electronics Engineering, IIT(BHU) Varanasi, for his valuable guidance, encouragement and help for accomplishing this work. His useful suggestions for this whole work and co-operative behaviour are sincerely acknowledged.

We would also like to express our sincere thanks to all others who helped us directly or indirectly during this project work.

Place: Varanasi **Students Name: -**

Date: 1-05-2020 Pranav Mittal -19095076

**Introduction**

As the number of applications of IOT and wireless communication arises, the need of encryption algorithms increases which are fast and secure (as the hardware has to limited) and also to remove errors during communication. Manufacturers know that encryption must be the silver bullet to protect their products and customers.

In short, AES is a **symmetric** type of encryption, as it uses the same key to both encrypt and decrypt data. It also uses the SPN (substitution permutation network) algorithm, applying multiple rounds to encrypt data. These encryption rounds are the reason behind the impenetrability of AES, as there are far too many rounds to break through.

To code, we have used a sample image that has been encoded to bitstream of 128 bit using python code. This bitstream is encoded using following modes.

1) Electronic Code Book Mode

2) Cipher Block Chaining Mode

3) Cipher Feedback Mode

4) Output Feedback Mode

Then it is decoded and the bits are converted to image using python code.

**Implementation**

Input is 1024x1024 grayscale image which has to be converted into bits and sent after implementing one of modes of AES Encryption technique.



First the image is converted to binary bit and saved in a .txt file that is read from vivid testbench and used as bit input for encoding and output for decoding processes. Also using python NumPy library we can convert the .txt bit file to image again.

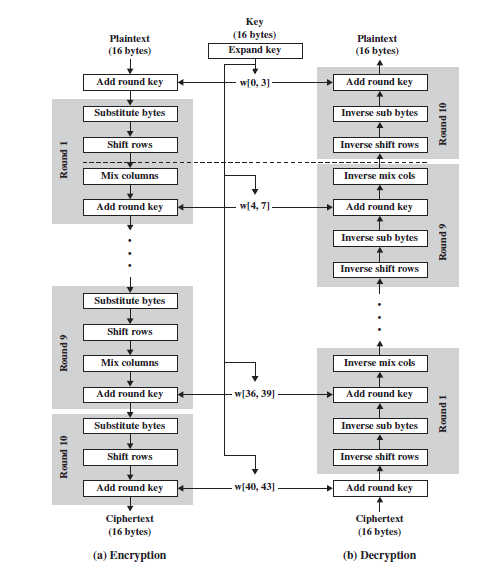
The AES encryption process[1]. The cipher takes a plaintext block size of 128 bits, or 16 bytes. And for the key length 16

bytes (128bits) the algorithm is referred to as AES-128.

AES structure

|  |  |
| --- | --- |
| The key | expanded into an array of forty-four 32-bit  words, **w**[*i*]. Four distinct words (128 bits) serve as a round key for each round |
| Four different stages are used, one of permutation and three of substitution | |
| **Substitute bytes:** | Uses an S-box to perform a byte-by-byte substitution of  the block |
| **ShiftRows:** | A simple permutation |
| **MixColumns:** | A substitution that makes use of arithmetic over |
| **AddRoundKey:** | A simple bitwise XOR of the current block with a portion  of the expanded key |

Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext.



**Substitute Bytes Transformation**

AES defines a matrix of byte values, called an S-box, that contains a permutation of all possible 256 8-bit values the 128-bit plaintext is converted to a 4x4 matrix where each element is of 8 bits or two hex numbers. Each individual byte of State is mapped into a new byte using the tables where first four bits represent the row and last four bits represent the column.

In Verilog, an array of registers is used to store the S box and inverse S box and based on the .8-bit text the new byte is returned via the function.

**ShiftRows Transformation**

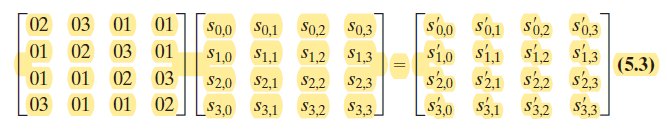
The matrix of 4x4 for forward transformation left shift is performed and for backward transformation right shift is performed. For first row no shift, for 2nd row 1 byte shift, for 3rd row 2-byte shift. so on.

In Verilog, this has been implemented by giving hard coded values between 1and 128, range of input.

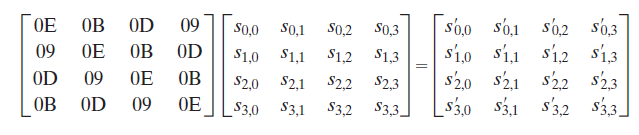
**MixColumns Transformation**

The matrix of 4x4 is transformed by performing array multiplication. All operations are performed on 8-bit bytes. In particular, the arithmetic operations of addition, multiplication, and division are performed over the finite field. So, each multiplication with a number represents a special operation[2]. After this operation addition is done i.e., XOR in GF.

For encryption.



For decryption



**AddRoundKey Transformation**

In the forward and backward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key. As XOR with same number two times returns the same number.

**Key Expansion Algorithm**

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the

cipher. The pseudocode on the next page describes the expansion.

|  |  |
| --- | --- |
| PSUEDO CODE (units in bytes) | Subworld Function |

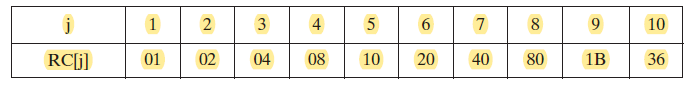
**1.** Rot Word performs a one-byte circular left shift on a word. This means that an

input word is transformed into.

**2.** SubWord performs a byte substitution on each byte of its input word, using the

S-box (Table 5.2a).

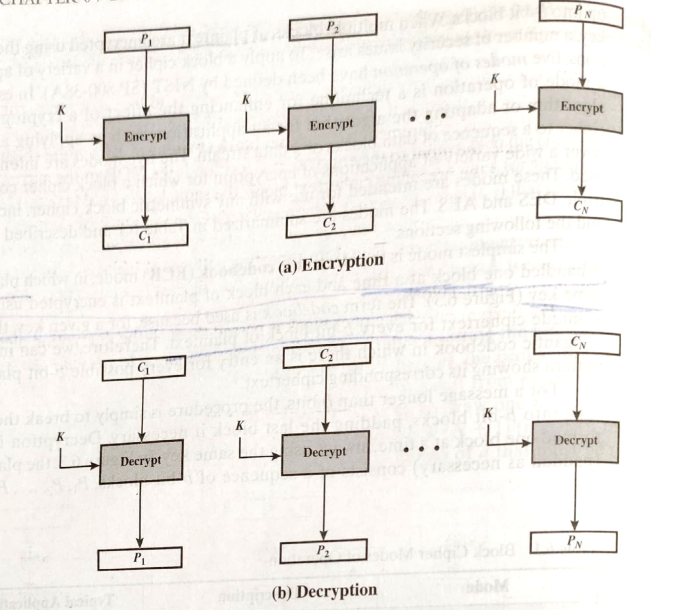
**3.** The result of steps 1 and 2 is XORed with a round constant.



All of these can be easily implemented in Verilog using functions.

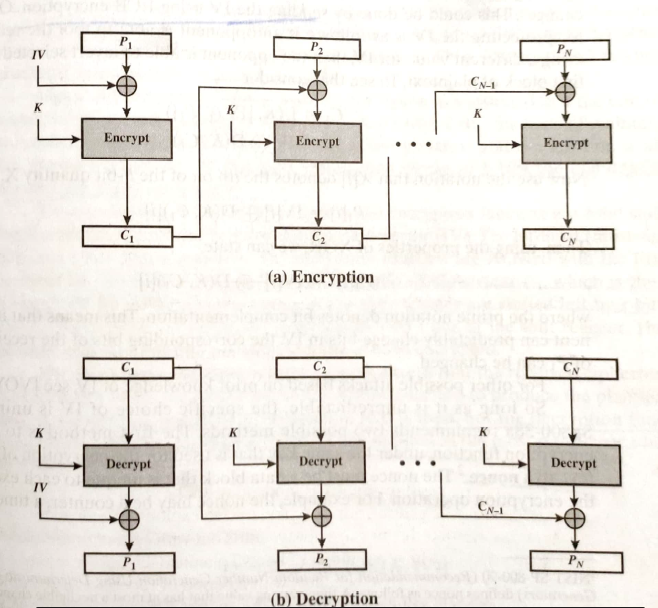
**ECB Mode or Electronic CodeBlock:**

A key is Xor-ed with text to generate cipher. Doing this again returns us the plain text.

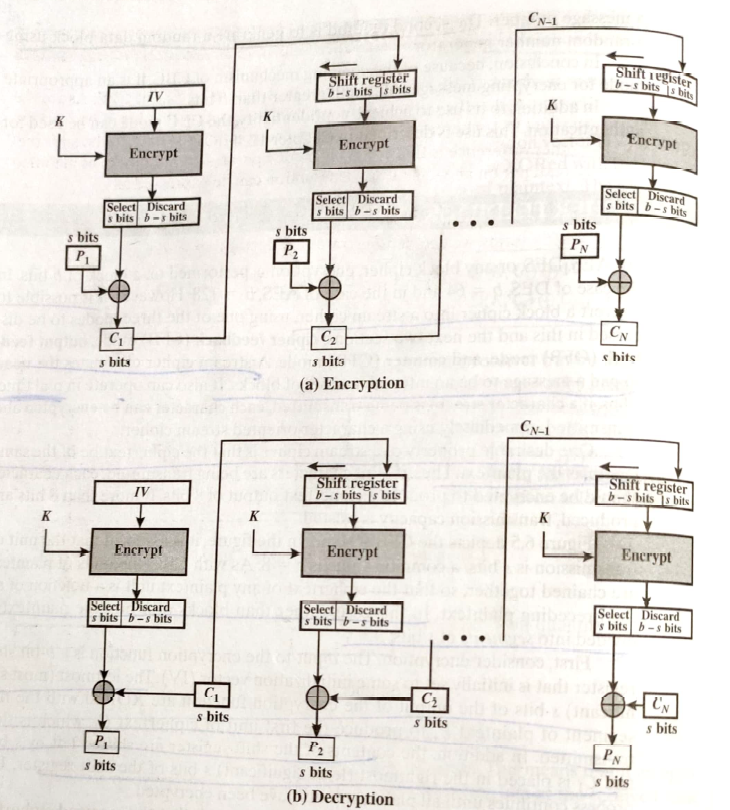


**CBC Mode or Cipher Block Chaining:**

Input to the encryption algo is the Xor of the next 128-bit plaintext and the preceding 128 bit of cipher text.

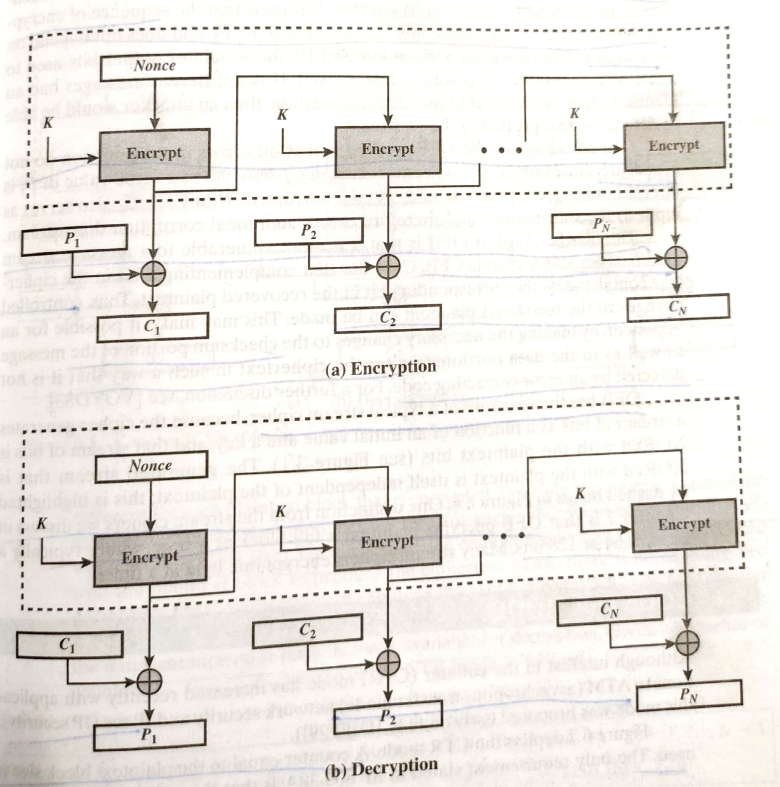


**CFB Mode Cipher CodeBlock:** Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is Xored with plaintext to produce next unit of ciphertext.



**OFB Mode or Output Feedback:**

Similar to CFB, except that input to encryption algorithm is the preceding encryption output, and full blocks are used.



**Conclusion**

The basic AES structure remains same in all the modes which makes the system very robust.

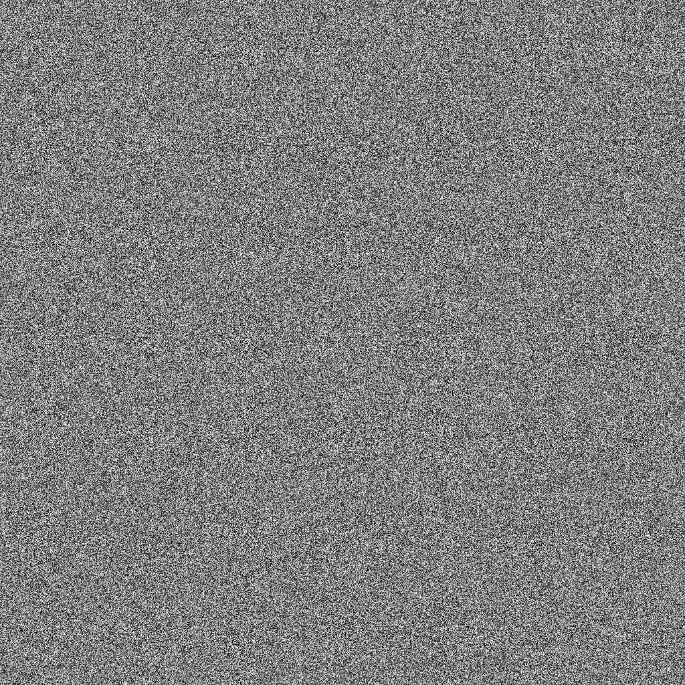
For lengthy message ECB is not secure. As if the message has repetitive elements an analyst can predict the possible word. It is good for transmitting encrypted key.

For CBC mode the input to encryption algorithm is the XOR of the current plain text block and preceding ciphertext block and the same key is used for each block, so the repeating patterns are not exposed.

The potential problem is that a bit error occurred during transmission will get propagated easily. Hence protective measures like crcgenerator[3] and retransmission[4] will have to be considered.

The CFB mode can operate in real time with s bit input. The advantage of OFB is that bit errors in transmission do not propagate. The disadvantage is that OFB vs CFB is that OPF is more vulnerable to message stream attack than in CFB. Complementing a bit in cipher text complements a bit in the recovered plaintext.

For wrong key, initial value the output plaintext / here the final image changes drastically.



All the given modes have advantages and their disadvantage as some are more secure and some might prove to be more power and hardware efficient. So, depending on the system either mode can be used or the encryption module needs to standardised for all systems for applications of IOT to increase.

For FPGA implementation all the components need to be perfectly timed and implementation has to be real time for its future applications to increase.

**Future Endeavours**

* To design RTL model for the code written and increasing the THROUGHPUT
* Using the same architecture for 64 bit and 32-bit cipher text.

**References**

[1] Chapter 5, Crytography And Network Security By William Stallings

[2]GALOIS FIELD MULTIPLICATION<https://arxiv.org/ftp/arxiv/papers/1209/1209.3061.pdf>

[3] https://www.geeksforgeeks.org/modulo-2-binary-division/

[4] https://en.wikipedia.org/wiki/OSI\_model