AES-128

AN IMPLEMENTATION IN SYSTEMC

ADITH M H

CONTENTS

ENCRYPTION AND DECRYPTION

ENCRYPTION

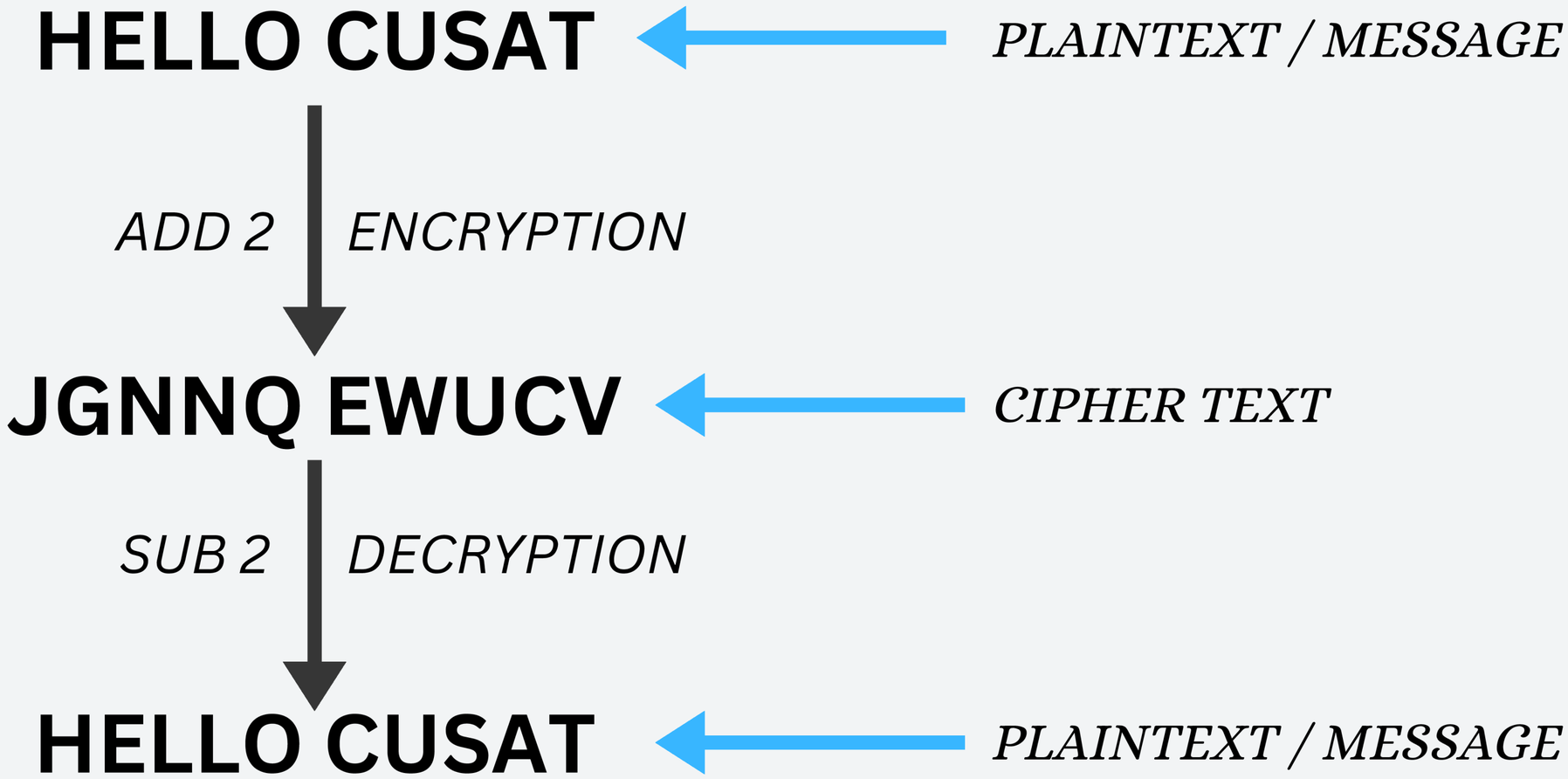
Process of converting readable or plaintext data into an unreadable or ciphertext format using an algorithm and an encryption key

SYMMETRIC
(SAME KEY)

ASYMMETRIC
(DIFFERENT KEY)

DECRYPTION

Process of converting ciphertext back into plaintext using the appropriate decryption key


HELLO CUSAT
PLAINTEXT / MESSAGE
ADD 2
ENCRYPTION
JGNNQ EWUCV
CIPHER TEXT
SUB 2
DECRYPTION
HELLO CUSAT
PLAINTEXT / MESSAGE

ADVANCED ENCRYPTION STANDARD (AES)

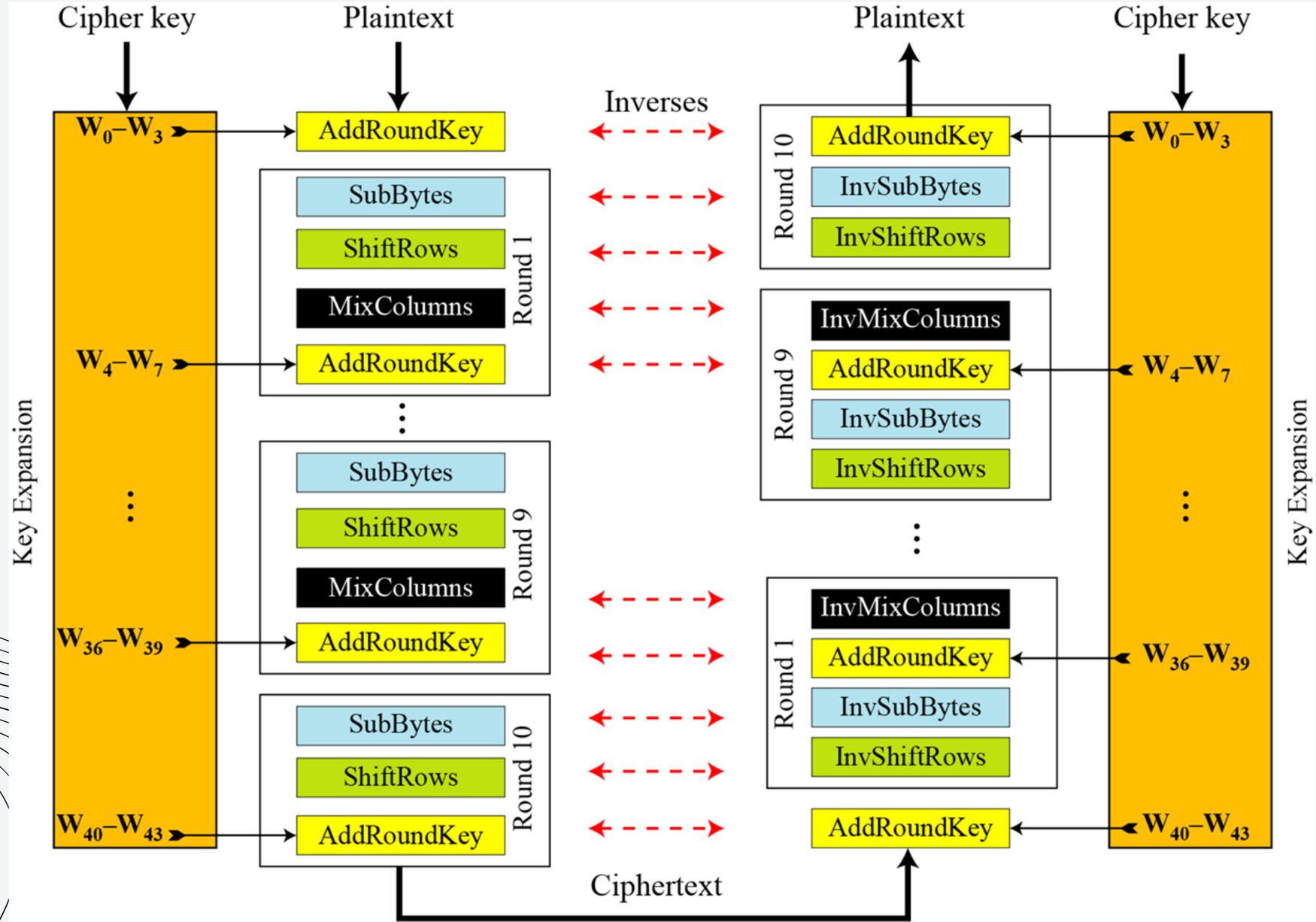
- Block cipher encryption algorithms
- Published by National Institute of Standards and technology (NIST) in 2001.
- Replaced the DES algorithm after some vulnerable aspects of it.

Algorithm	Key Length (Bits)	Block Size (Bits)	No. of Rounds
AES-128	128	128	10
AES-192	192	192	12
AES-256	256	256	14

APPLICATIONS

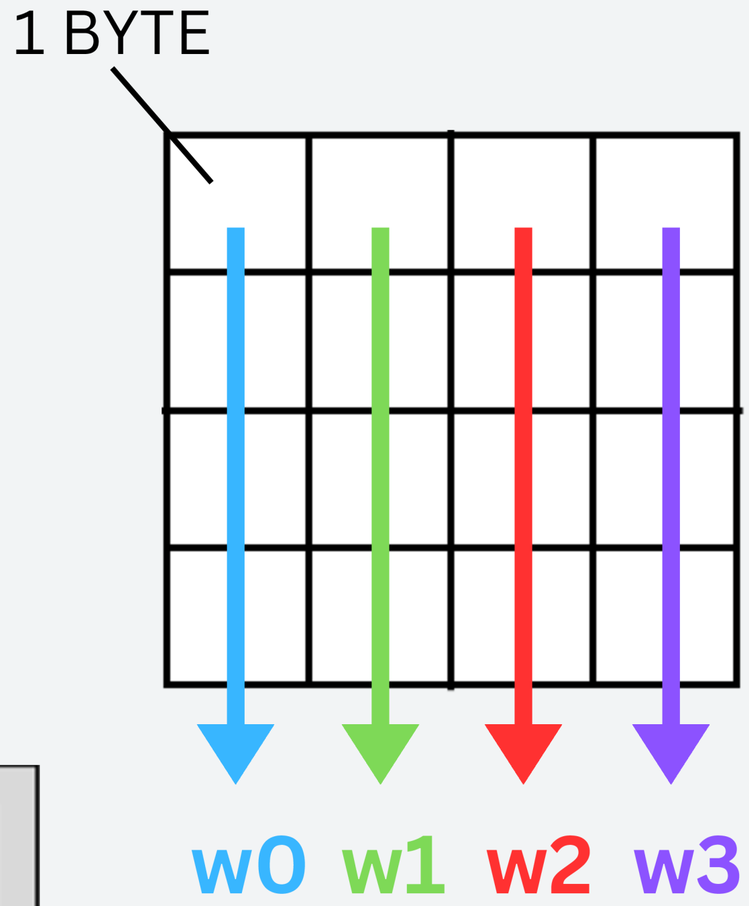
- COMPUTERS AND LAPTOPS
- MOBILE DEVICES
- WiFi
- PAYMENTS SYSTEMS
- NETWORKING DEVICES
- SOFTWARE APPLICATIONS

STRUCTURE OF AES-128



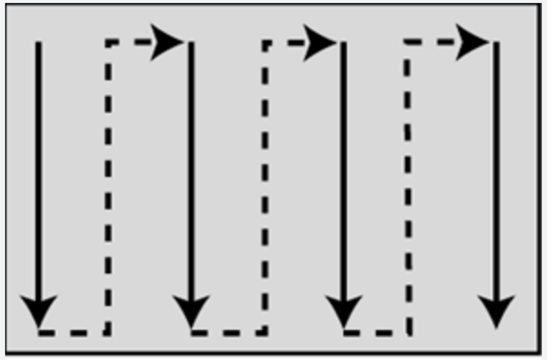
128 BIT VALUE

- DATA BLOCK VIEWED AS 4-BY-4 TABLE OF BYTES
- REPRESENTED AS 4 BY 4 MATRIX OF 8-BIT BYTES.
- KEY IS EXPANDED TO ARRAY OF 32 BITS WORDS

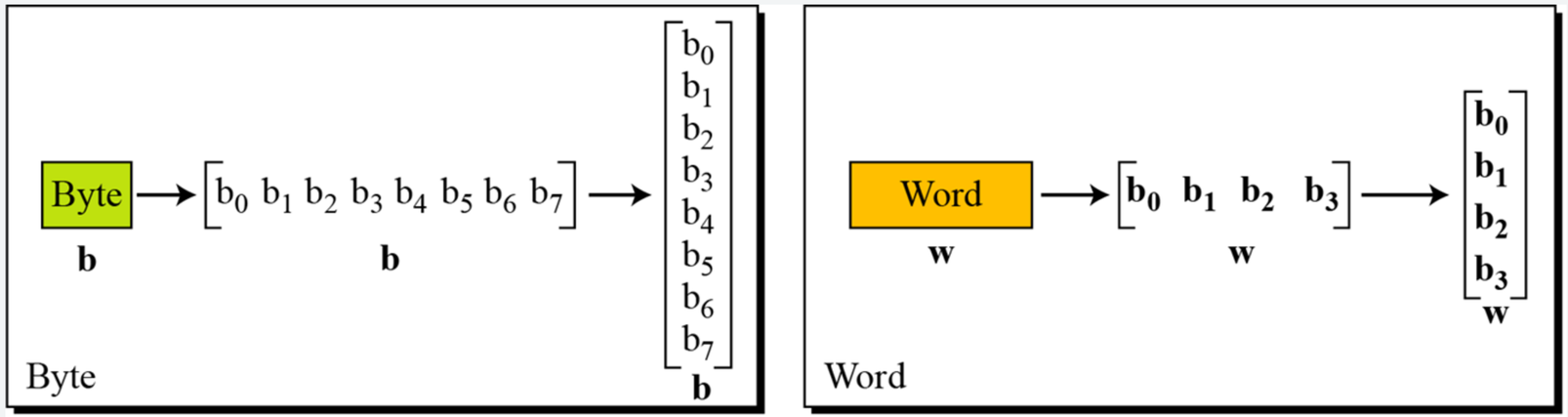


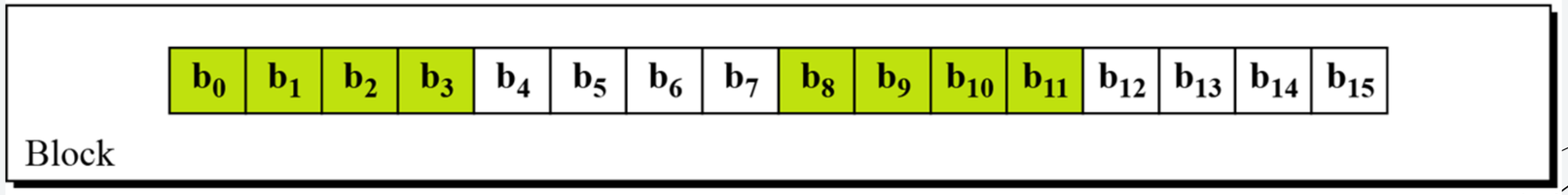
1 byte = 8 bit

1 word = 4 byte = 32 bit



Insertion and extraction flow





Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

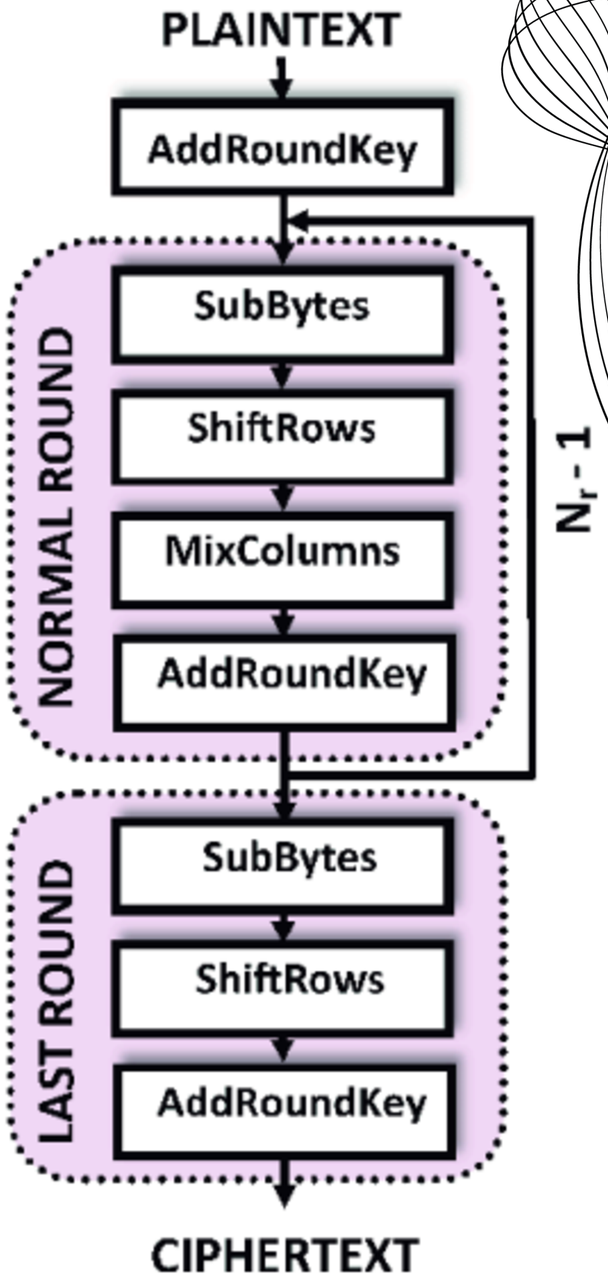
$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$ State

AES ENCRYPTION

A single round contains :

1. **SubBytes**
2. **ShiftRows**
3. **MixColumns**
4. **AddRoundkey**

LAST ROUND DOES NOT CONTAIN MIX COLUMN

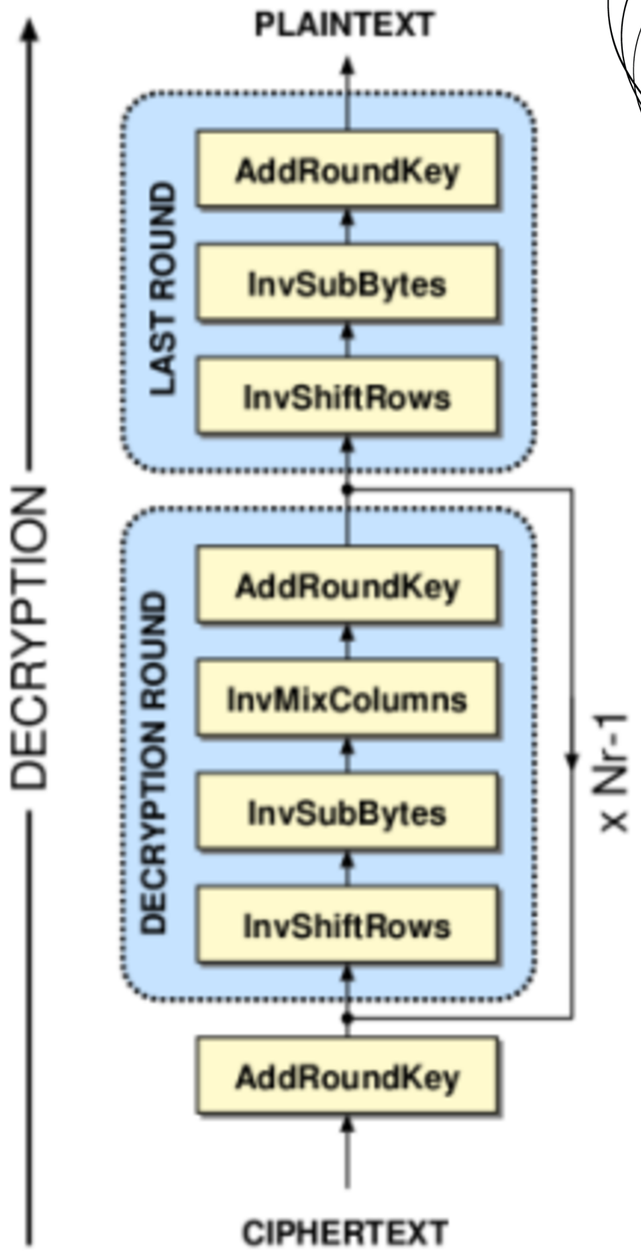


AES DECRYPTION

A single round contains :

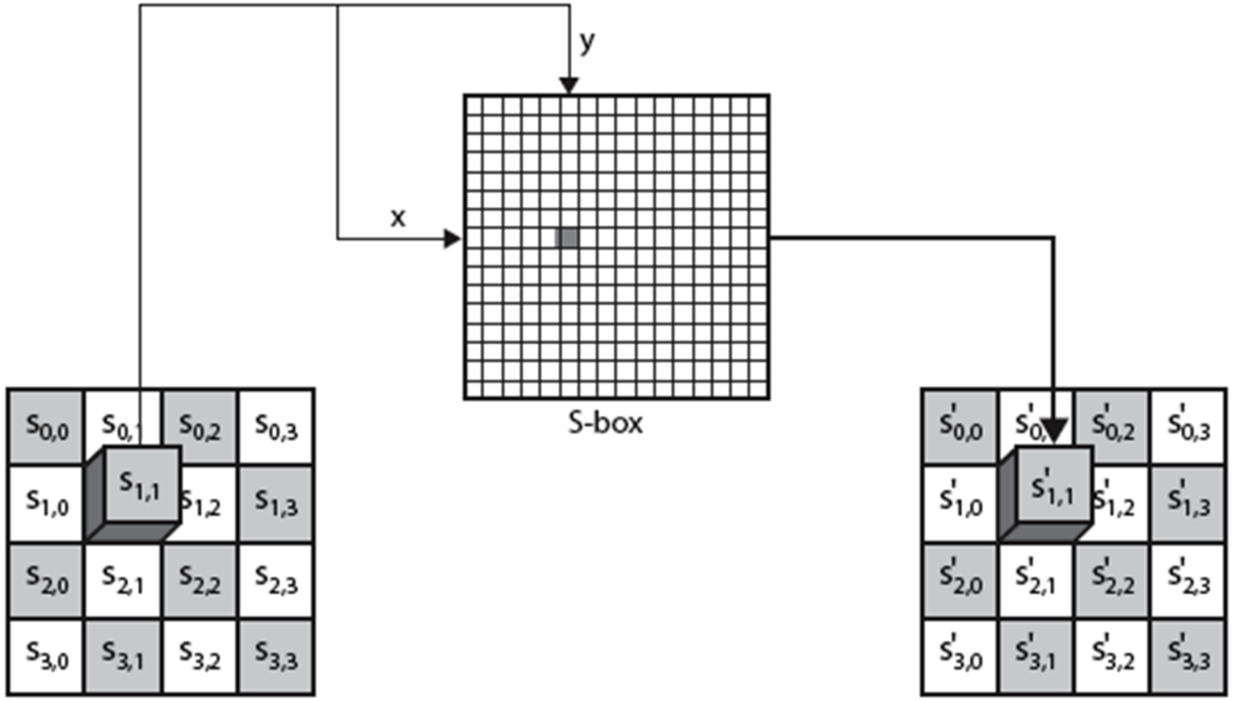
1. **InvShiftRows**
2. **InvSubBytes**
3. **InvMixColumns**
4. **AddRoundkey**

LAST ROUND DOES NOT CONTAIN INVMIXCOLUMNS



SubBytes

- A simple substitutionof each byte
  - Provide a confusion
- Uses one S-Box of 16x16 bytes containing permutation of all 256 8-bit values
- Each byte of state is replaced by byte indexed row (left 4 bits) & column (right 4 bits)
  - eg : byte 95 is replaced by byte in row 9 and column 5
  - which has value 2A



S-Box

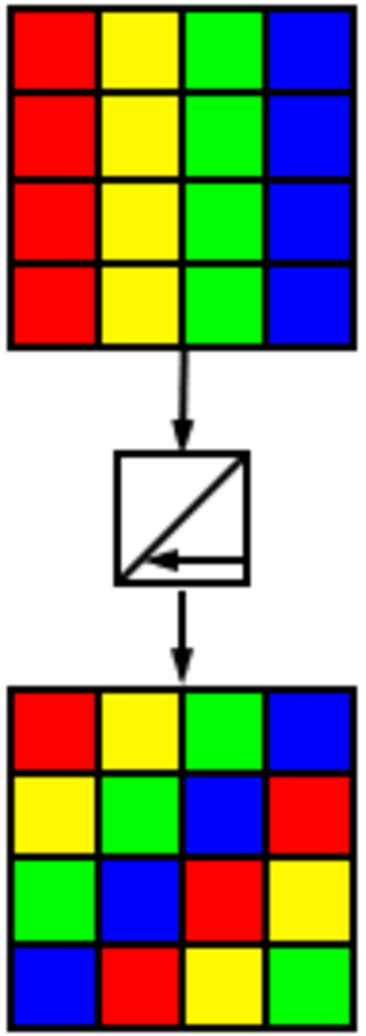
		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
x	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	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

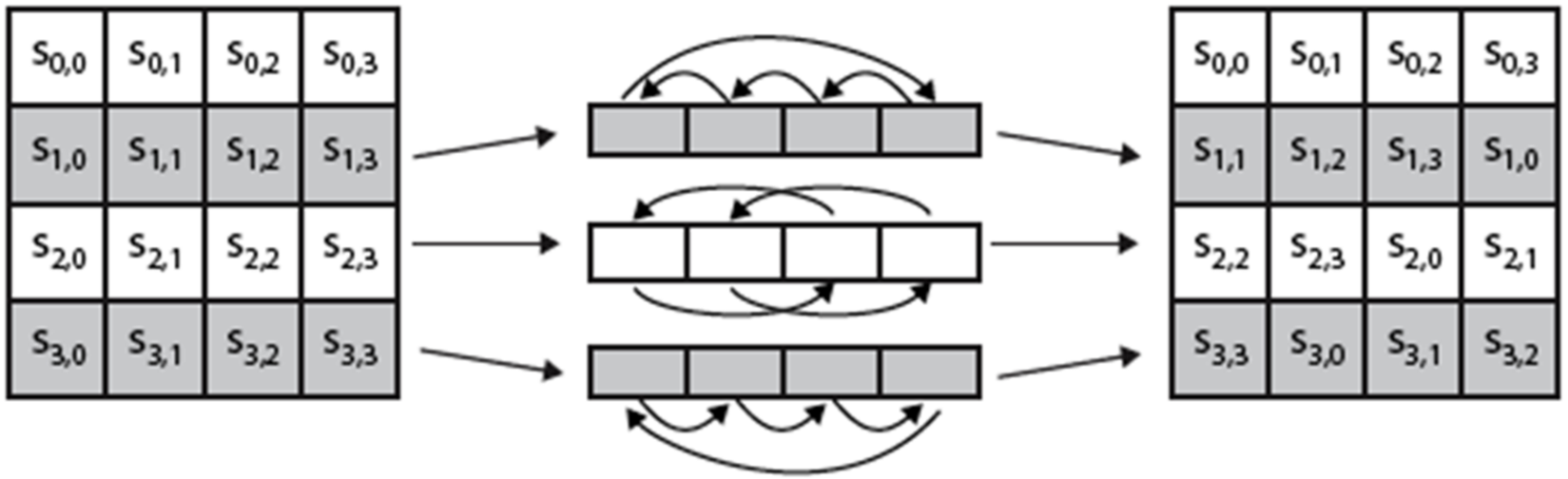
Inverse S-Box

		y														
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
x	**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

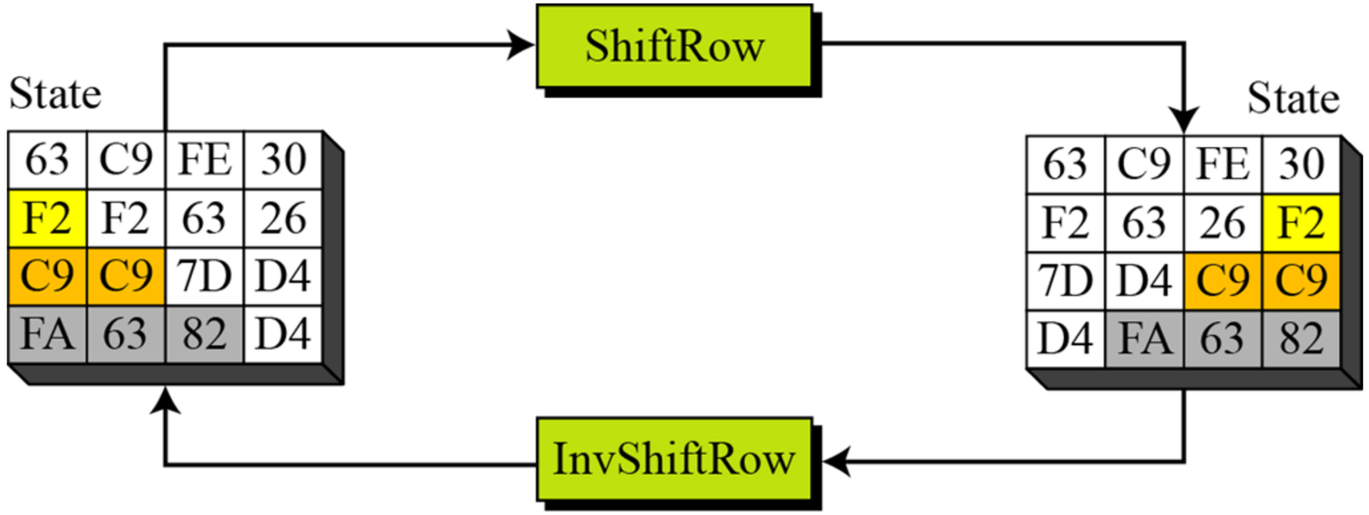
ShiftRows

- Shifting, which permutes the bytes.
- A circular byte shift
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- In the encryption, the transformation is called ShiftRows
- In the decryption, the transformation is called InvShiftRows and the shifting is to the right



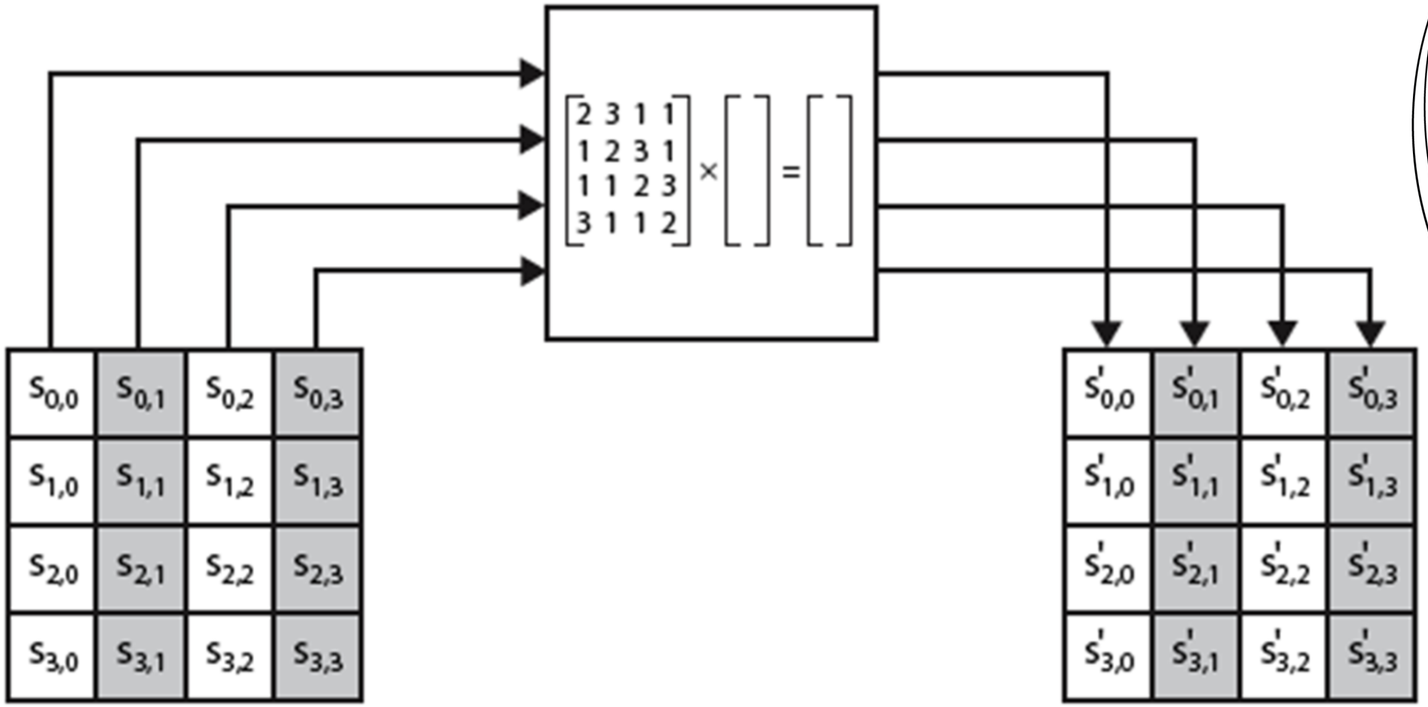


In InvShiftRows the rows are cyclically shifted to the Right

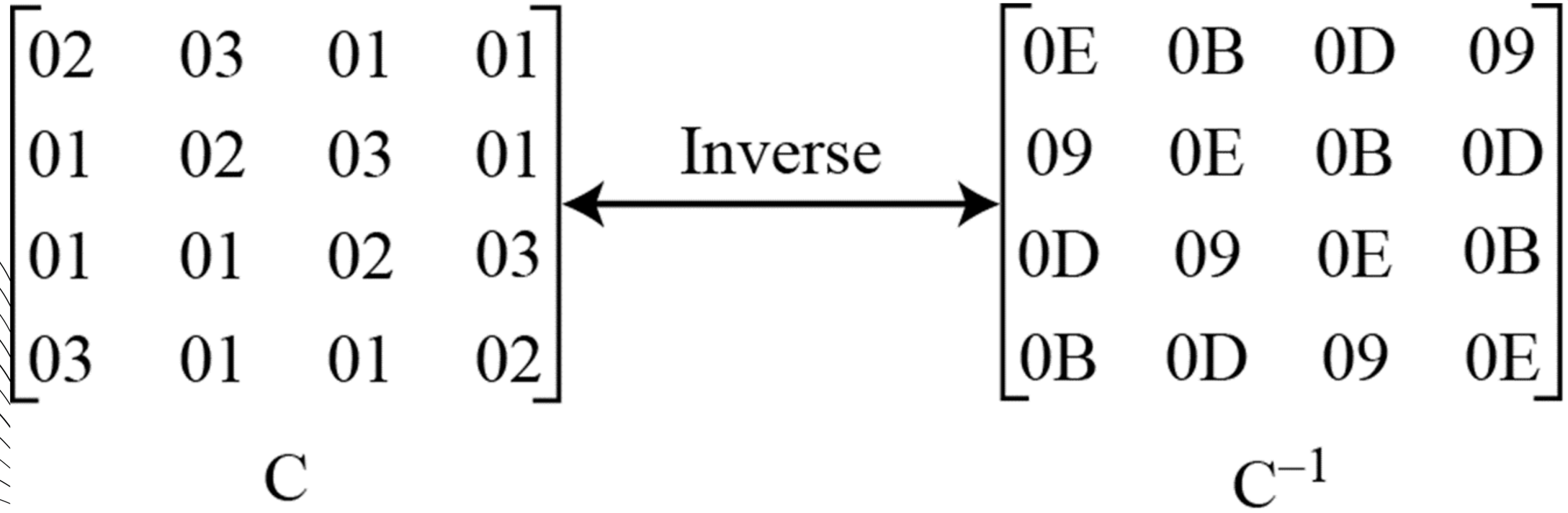


MixColumns

- ShiftRows and MixColumns provide diffusion to the cipher
- Each column is processed separately
- Each byte is replaced by a value dependent on all 4 bytes in the column

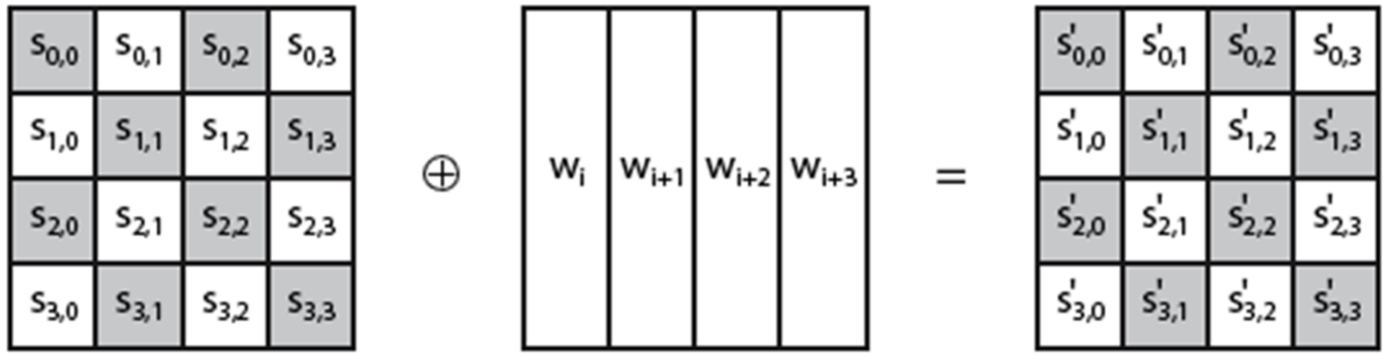


MixColumn and InvMixColumn



AddRoundKey

- XOR state with 128-bits of the round key
- AddRoundKey proceeds one column at a time.
  - Adds a round key word with each state column matrix
  - The operation is matrix addition
- Inverse for decryption identical
- Since XOR own inverse, with reversed keys
- Designed to be as simple as possible



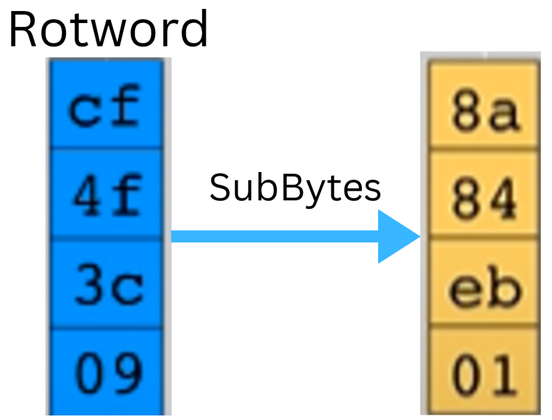
KEY EXPANSION

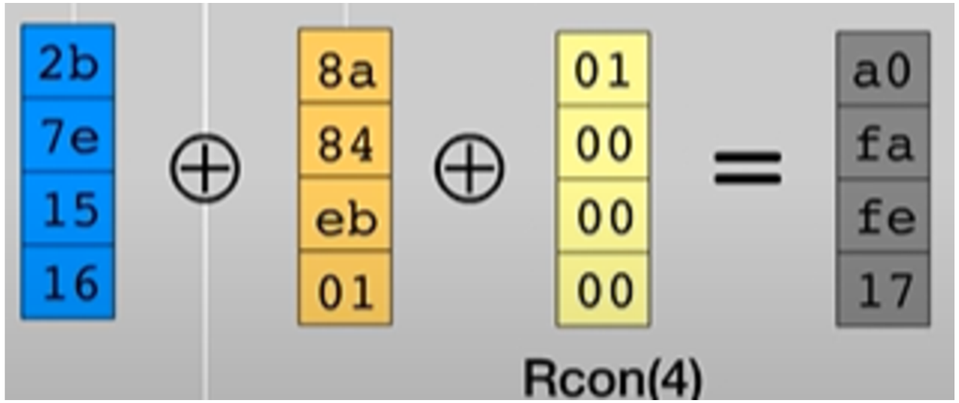
Wi-4	Wi-3	Wi-2	Wi-1	Wi				Wi+1			
2b	28	ab	09								
7e	ae	f7	cf								
15	d2	15	4f								
d2	a6	88	3c								

INITIAL KEY **ROUND 1 KEY** **ROUND 2 KEY**

To get Wi

- Take column Wi-1 and rotate it once circularly top to bottom , it is the Rotword
- Find the subbytes of rotword
- Do XOR operation of the Subbytes of Rotword and Wi-4 and First column of Rcon matrix





To get Wi+1

- Do XOR of Wi and Wi-3

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon Matrix

To get Wi+2

- Do XOR of Wi+1 and Wi-2

To get Wi+2

- Do XOR of Wi+2 and Wi-1

FOR GETTING THE FIRST COLUMN OF ROUND KEY 2 WE HAVE TO FOLLOW THE STEPS FOR GETTING Wi

REPEAT THE STEPS TO GET ALL ROUND KEY VALUES

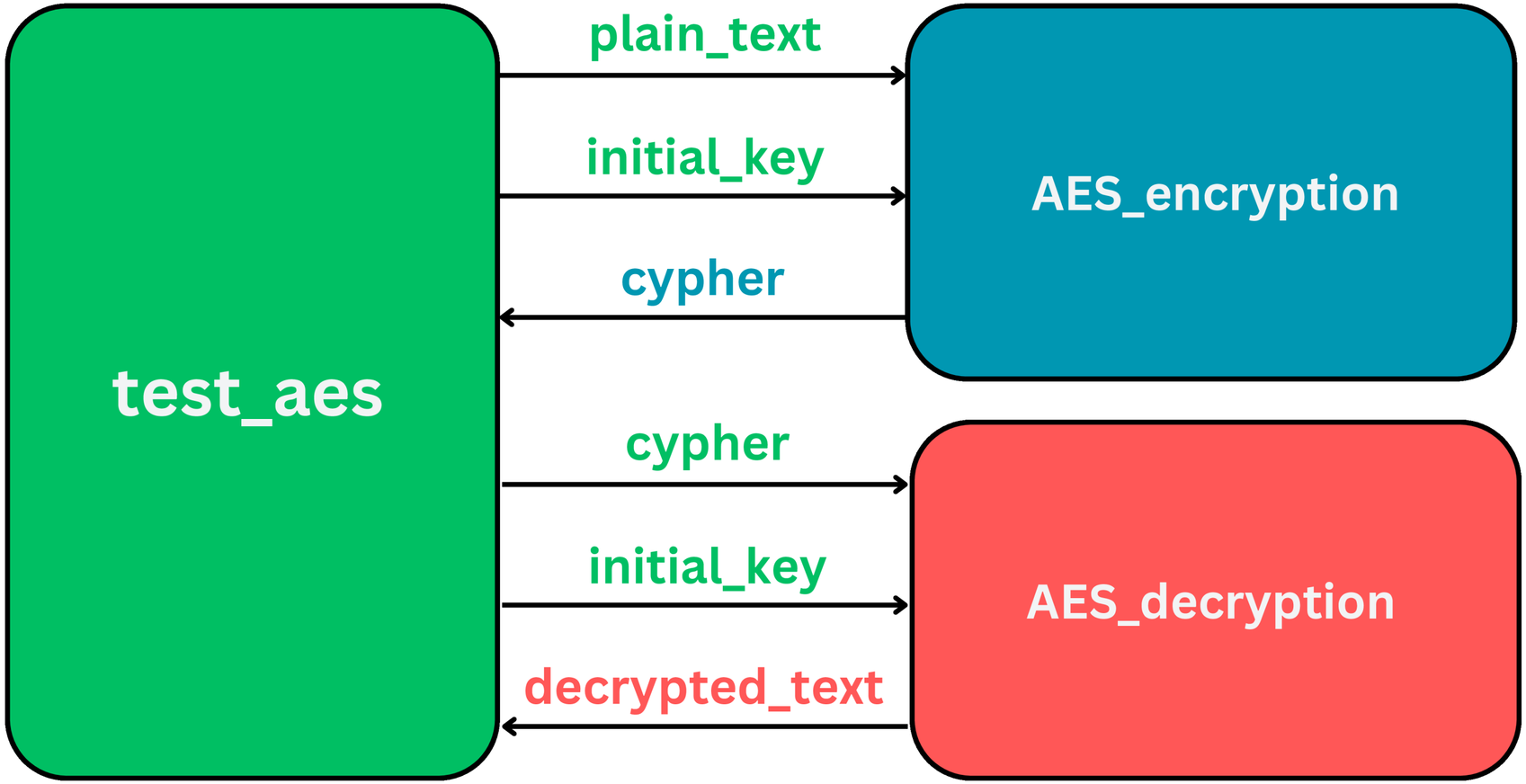
2b	28	ab	09	a0	88	23	2a	f2	7a	59	73	3d	47	1e	6d
7e	ae	f7	cf	fa	54	a3	6c	c2	96	35	59	80	16	23	7a
15	d2	15	4f	fe	2c	39	76	95	b9	80	f6	47	fe	7e	88
16	a6	88	3c	17	b1	39	05	f2	43	7a	7f	7d	3e	44	3b
Cipher key				Round key 1				Round key 2				Round key 3			

d0	c9	e1	b6	
14	ee	3f	63	
f9	25	0c	0c	
a8	89	c8	a6	
				Round key 10

SYSTEMC MODEL

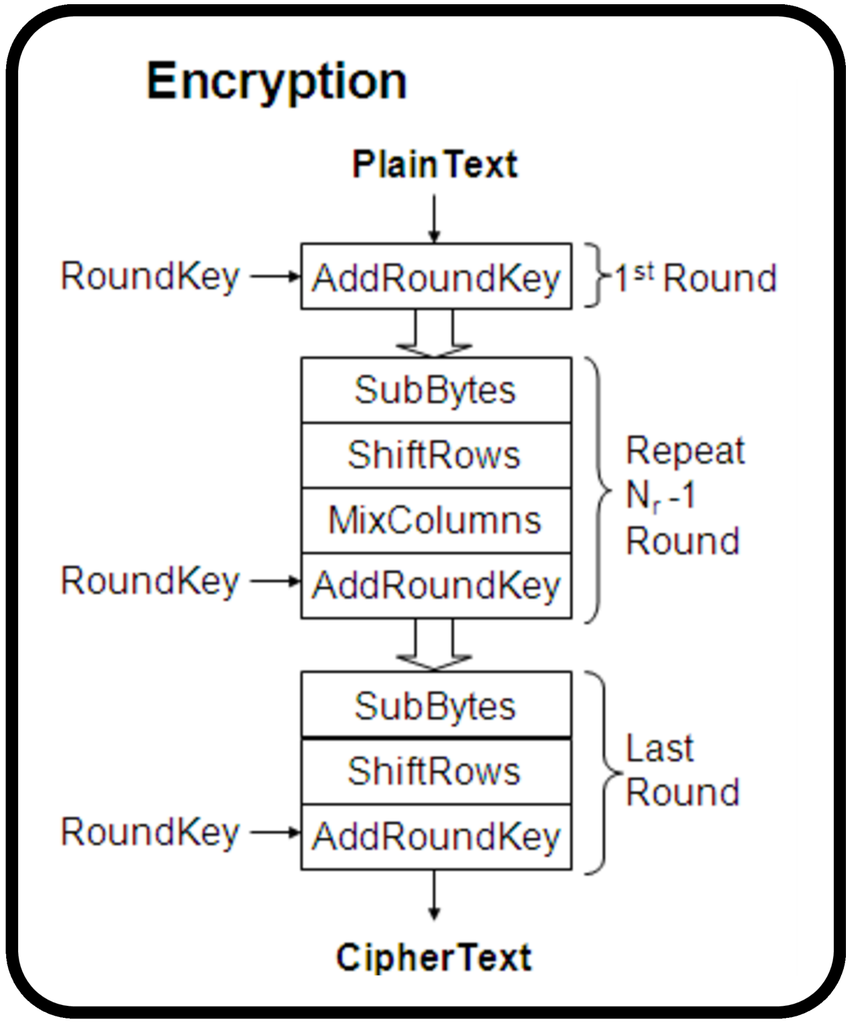
AES-128

Design of AES-128



All the inputs and outputs are 128 bit size

AES_encryption Module



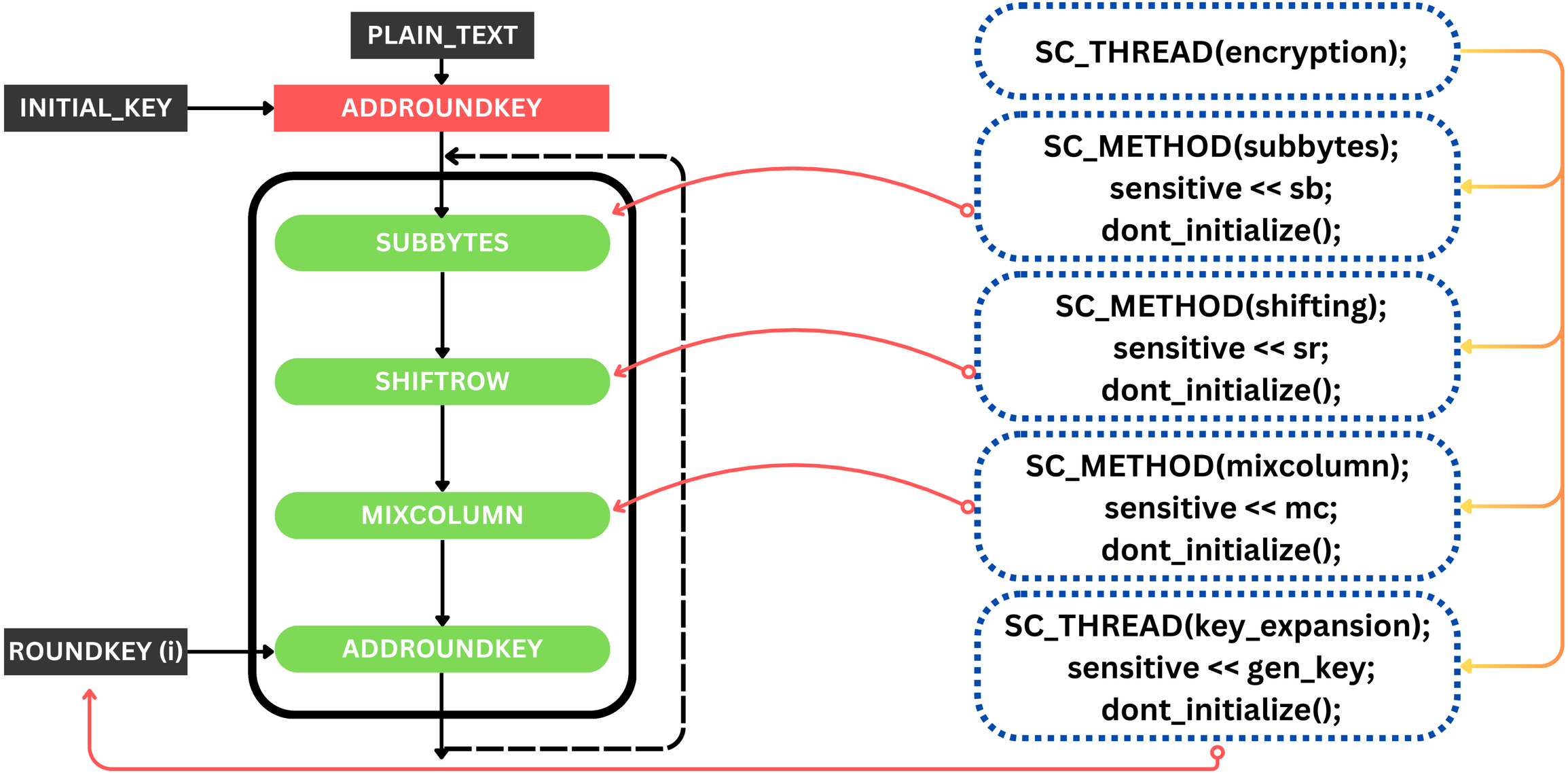
Plain Text : sc_biguint<128> plain_text

Initial key : sc_biguint<128> initial_key

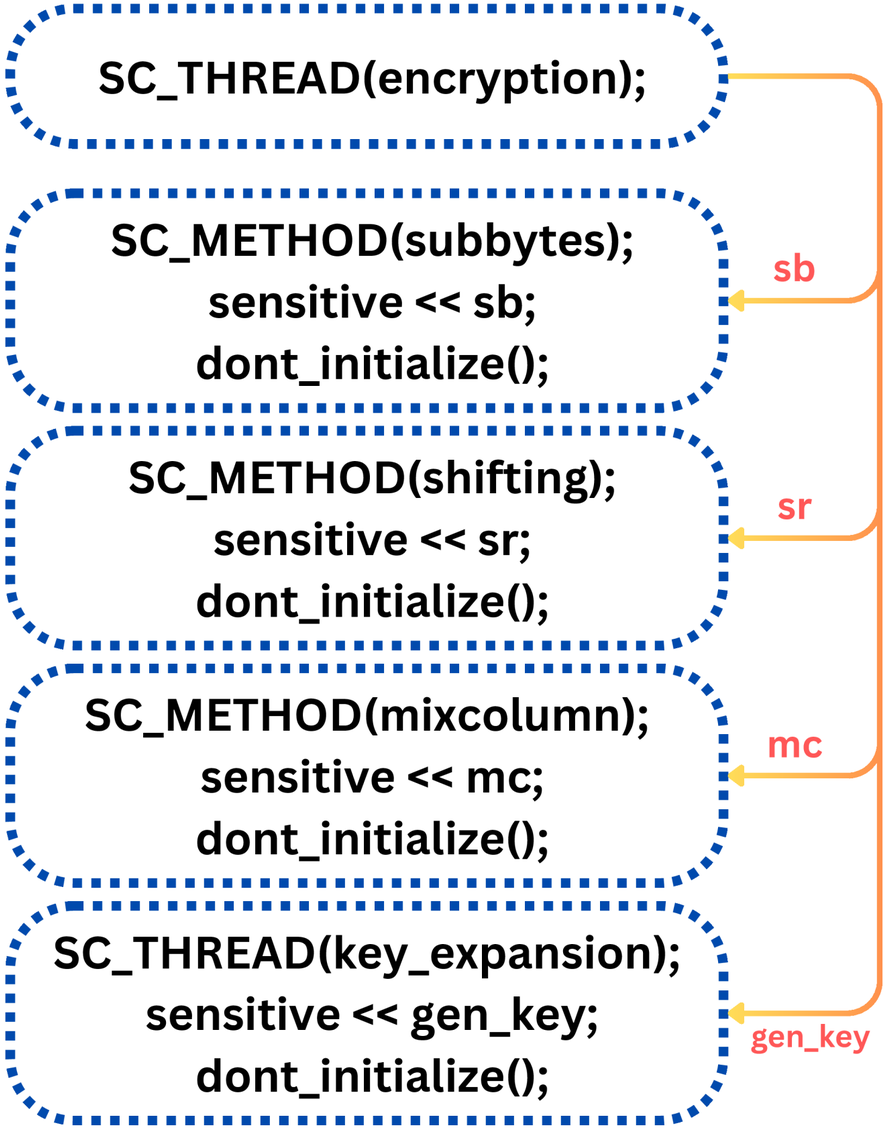
Cypher Text : sc_biguint<128> cypher

All the variables and signals are of the datatype sc_biguint<128>

Processes and Events in Encryption



Processes and Events in Encryption

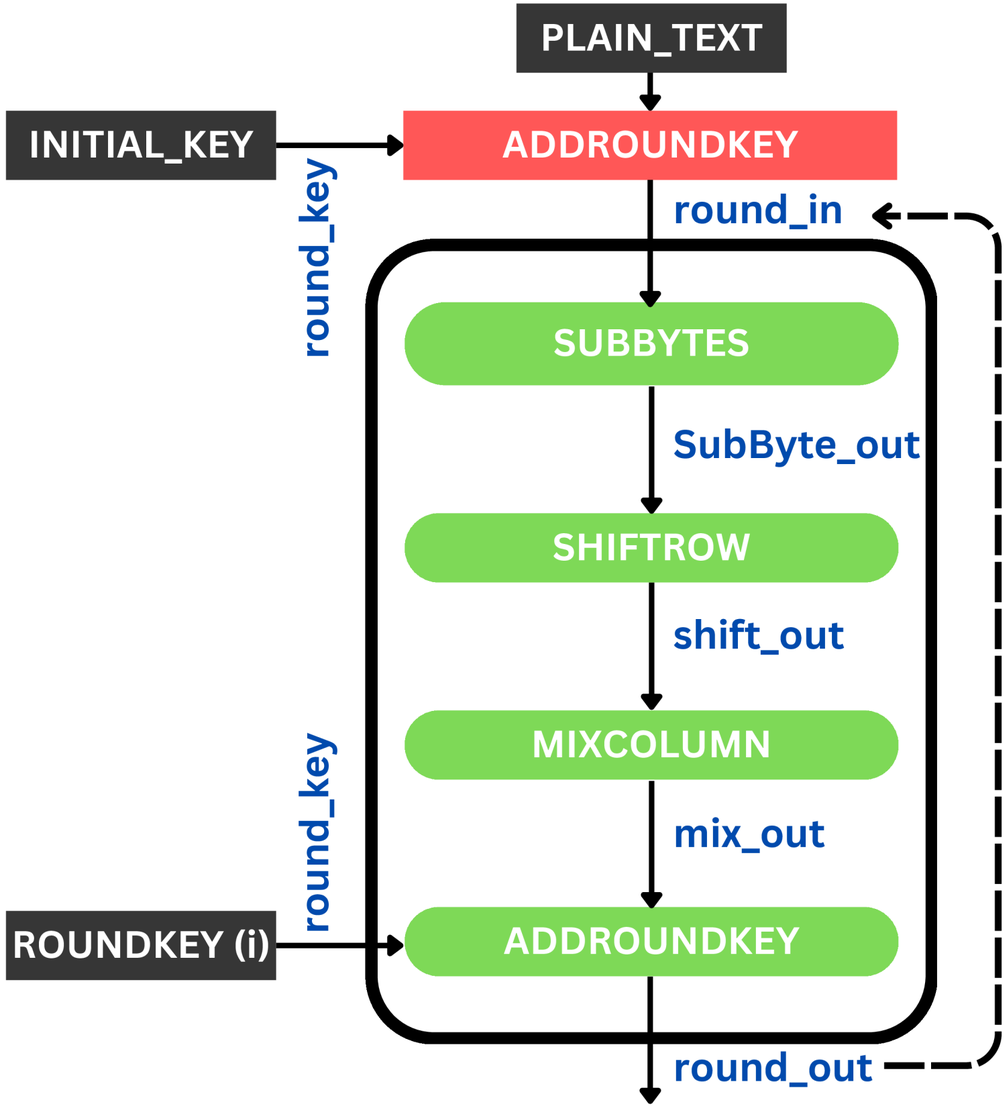



```
sc_event sb, sr, mc, gen_key,key_ready;
```

SC_THREAD(encryption)	• Implements the main encryption logic sequentially. • Waits for event key_ready just before the Addroundkey for the key to be ready for XORing
SC_METHOD(subbytes)	• Performs byte substitution using the S-Box. • Triggered by the sb event, ensuring execution only when required.
SC_METHOD(shifting)	• Executes the ShiftRows transformation for row-wise permutation. • Sensitive to the sr event, activating when ShiftRows is required.
SC_METHOD(mixcolumn)	• Handles MixColumns transformation for column-wise mixing. • Triggered by the mc event for selective operation.
SC_THREAD(key_expansion)	• Dynamically generates round keys for AES encryption. • Sensitive to the gen_key event, aligning with the need for new keys.

dont_initialize()	Ensures processes are not triggered during initialization but only when sensitive events occur.

Internal Signals in Encryption

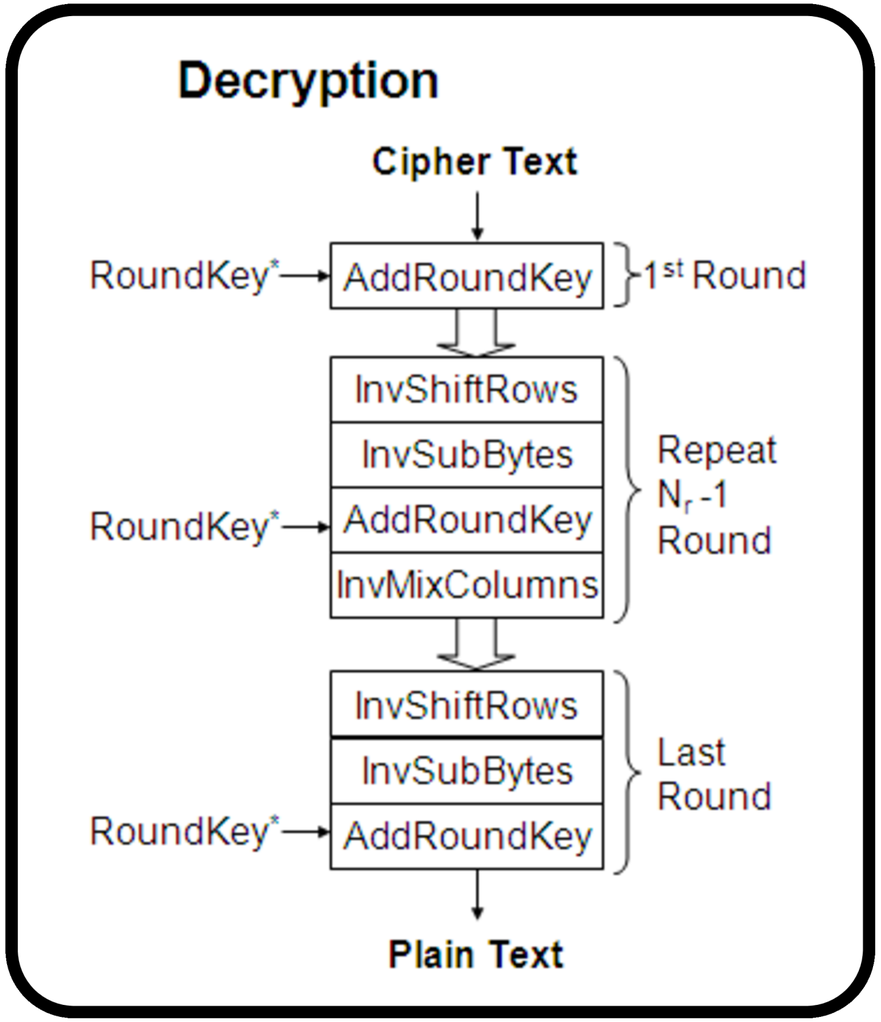


Signals Used

```
sc_signal<sc_biguint<AES_SIZE>> round_in;
sc_signal<sc_biguint<AES_SIZE>> SubByte_out;
sc_signal<sc_biguint<AES_SIZE>> shift_out;
sc_signal<sc_biguint<AES_SIZE>> mix_out;
sc_signal<sc_biguint<AES_SIZE>> round_out;
sc_signal<sc_biguint<AES_SIZE>, SC_MANY_WRITERS> round_key;
```

- **round_in**: Input to the current round of AES encryption.
- **SubByte_out**: Output of the SubBytes transformation.
- **shift_out**: Output of the ShiftRows transformation.
- **mix_out**: Output of the MixColumns transformation.
- **round_out**: Output of the current round of encryption.
- **round_key**: Current round key used in encryption (with multiple writers).

AES_decryption Module



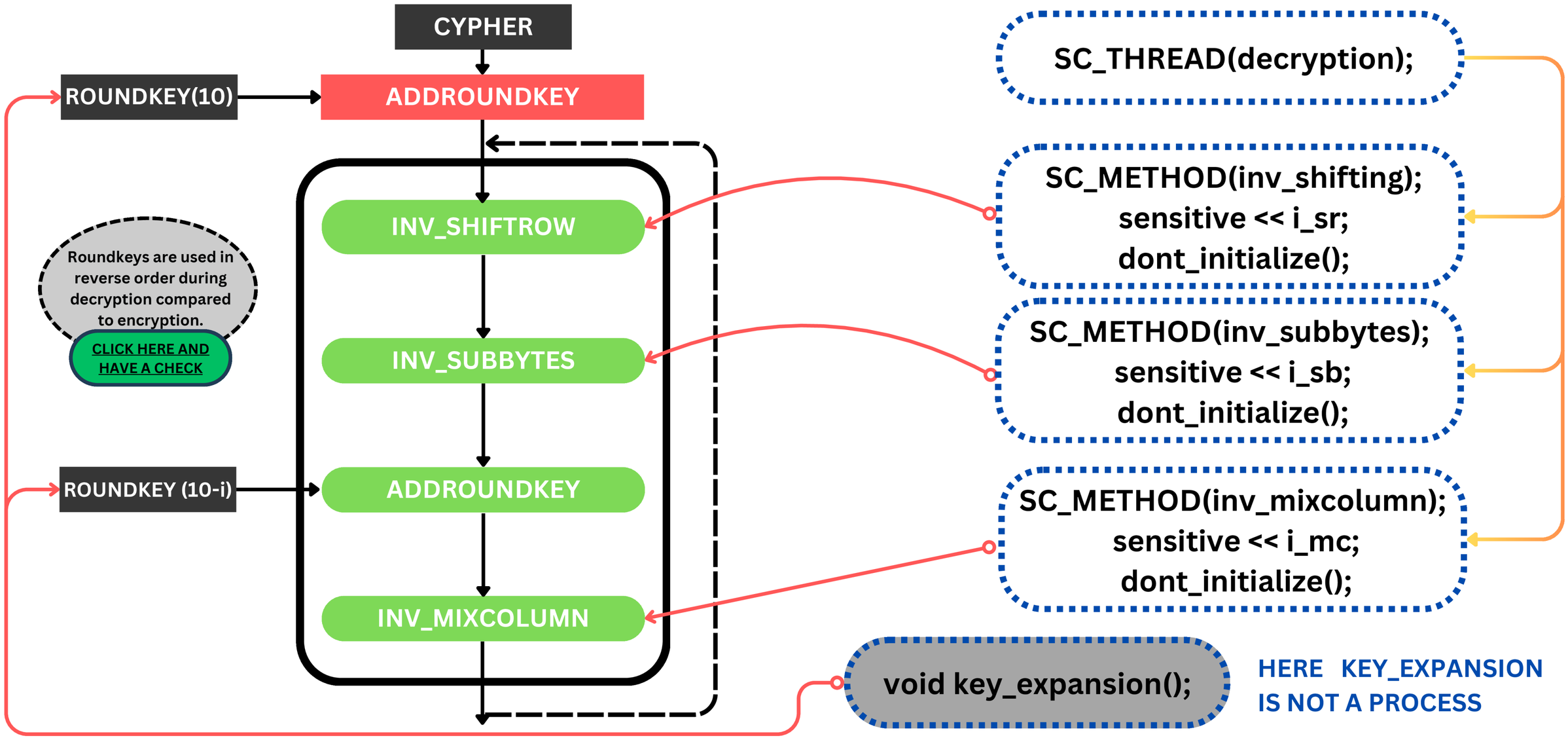
Cypher Text : **sc_biguint<128> cypher**

Secret key : **sc_biguint<128> secret_key**

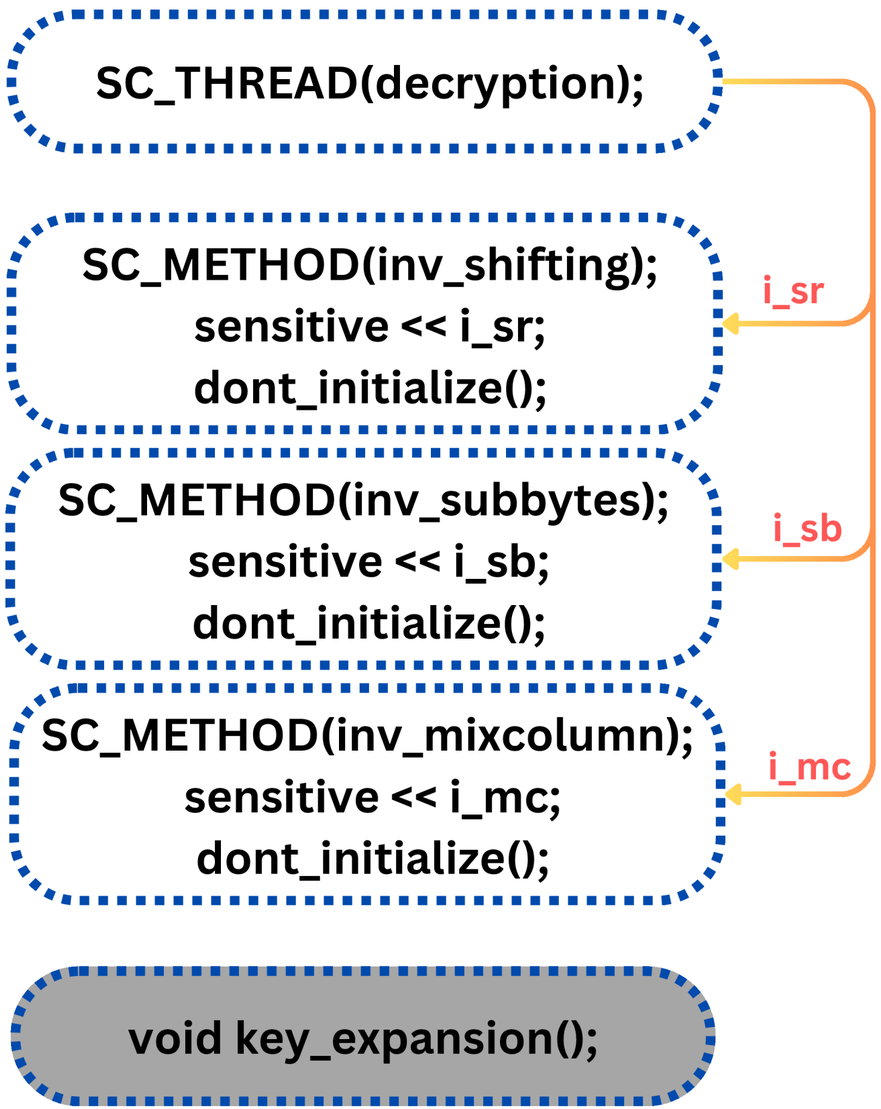
Decrypted text : **sc_biguint<128> decrypted_text**

All the variables and signals are of the datatype sc_biguint<128>

Processes and Events in Decryption



Processes and Events in Decryption



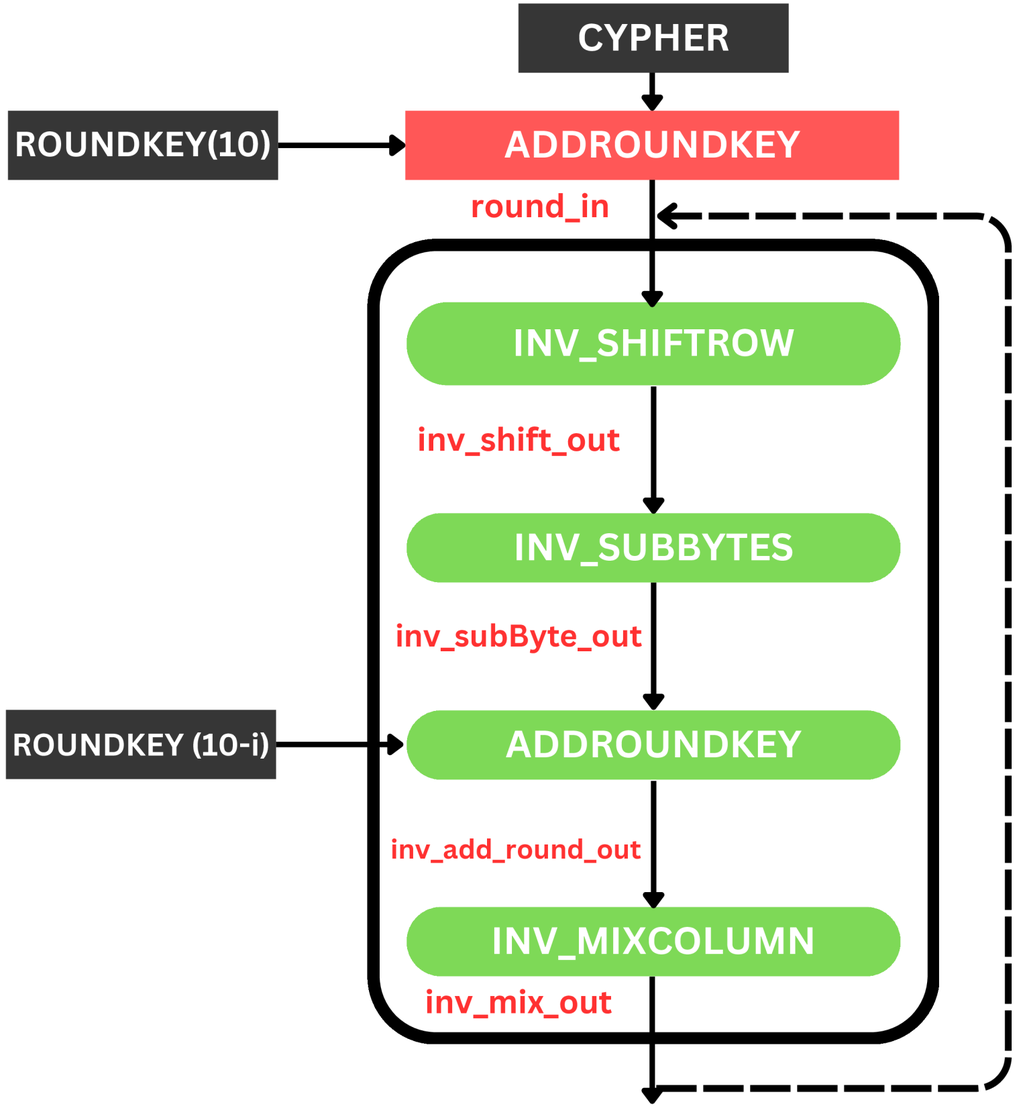

```
sc_event i_sb, i_sr, i_mc;
```

SC_THREAD(decryption);	• Implements the main decryption logic sequentially.
SC_METHOD(inv_shifting);	• Executes the inverse ShiftRows transformation for row-wise permutation. • Sensitive to the i_sr event, activating when inverse ShiftRows is required.
SC_METHOD(inv_subbytes);	• Performs byte inverse substitution using the S-Box. • Triggered by the i_sb event, ensuring execution only when required.
SC_METHOD(inv_mixcolumn);	• Handles inverse MixColumns transformation for column-wise mixing. • Triggered by the i_mc event for selective operation.

In decryprtion , key_expansion() is not a process, its just a member function

void key_expansion();	• Generates all round keys for AES decryption when called. • Store all the round keys to a vector called Round_keys ◦ std ::vector<sc_biguint<AES_SIZE>> Round_keys;

Internal Signals in Decryption

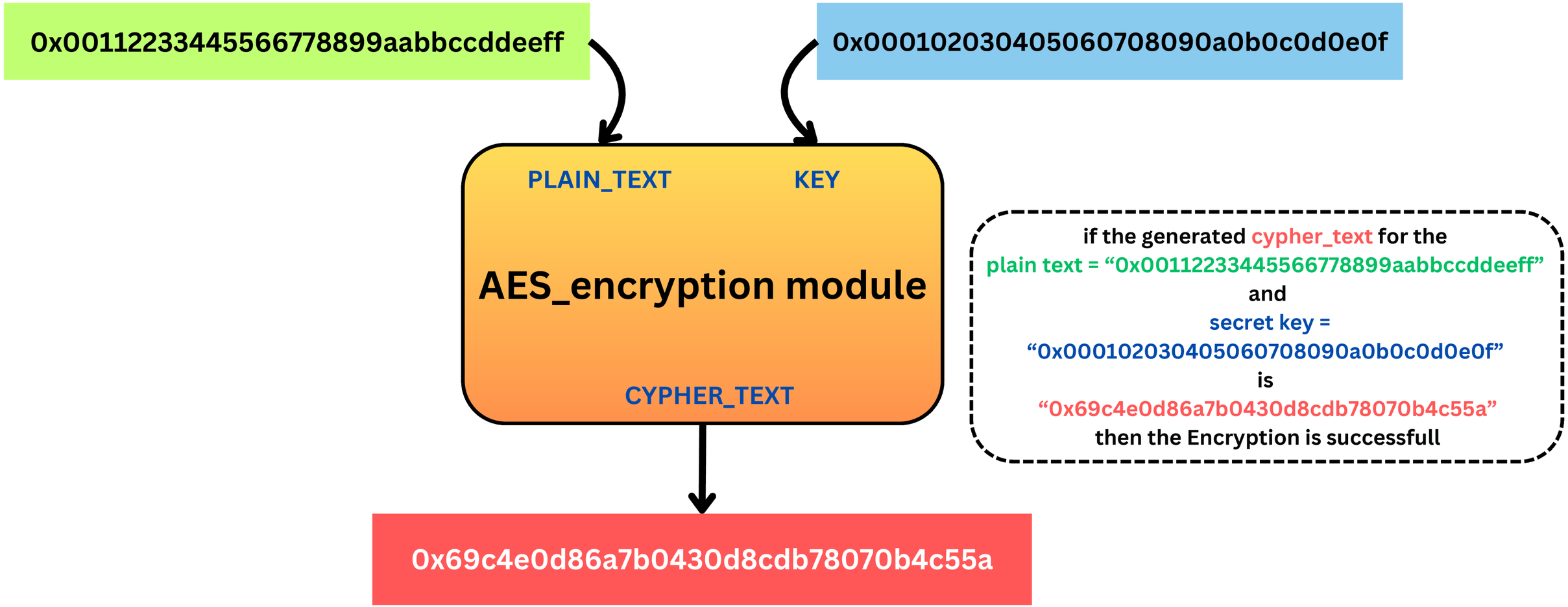


Signals Used

```
sc_signal<sc_biguint<AES_SIZE>> round_in;
sc_signal<sc_biguint<AES_SIZE>> inv_shift_out;
sc_signal<sc_biguint<AES_SIZE>> inv_subByte_out;
sc_signal<sc_biguint<AES_SIZE>> inv_add_round_out;
sc_signal<sc_biguint<AES_SIZE>> inv_mix_out;
```

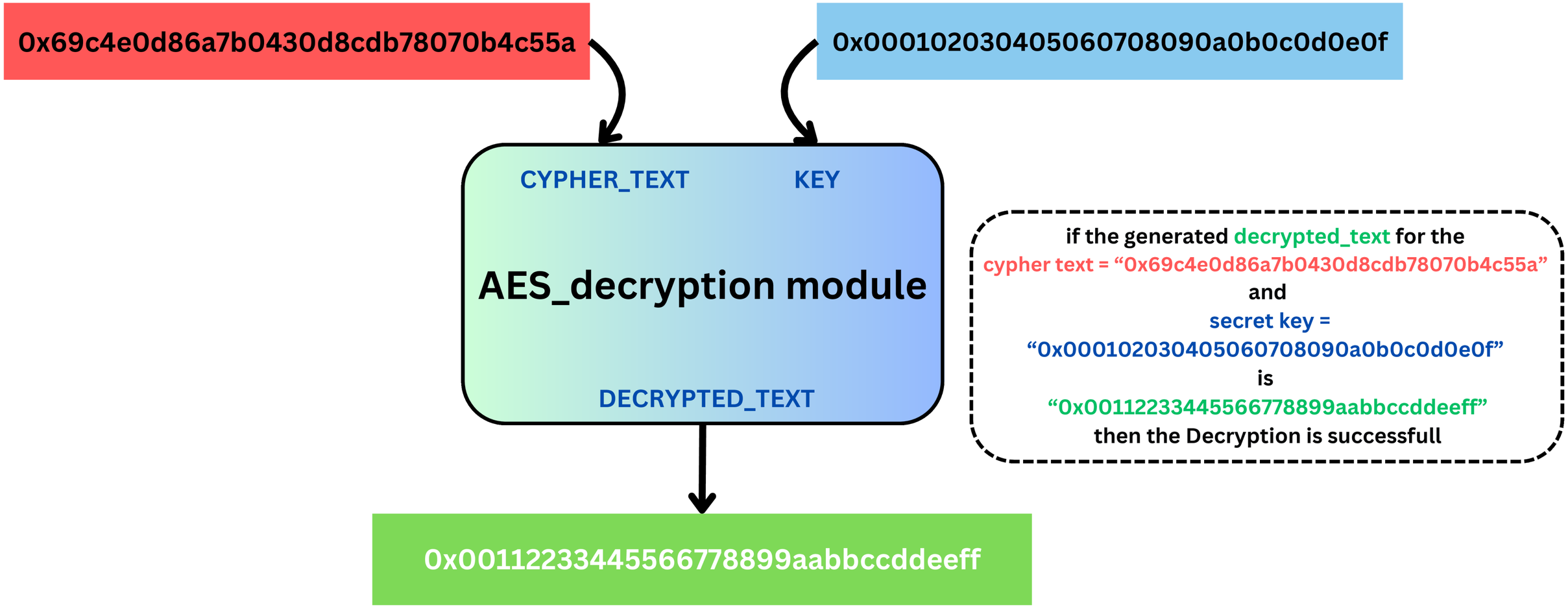
- **round_in**: Input to the current round of AES encryption.
- **inv_shift_out**: Output of the inverse shiftrow transformation.
- **inv_subByte_out**: Output of the inverse subbytes transformation.
- **inv_add_round_out**: Output of the add_round_key. This will be the round output for last round
- **inv_mix_out**: Output of the inverse mixcolumn transformation. This will be the round output for all rounds except last round

Testing Encryption

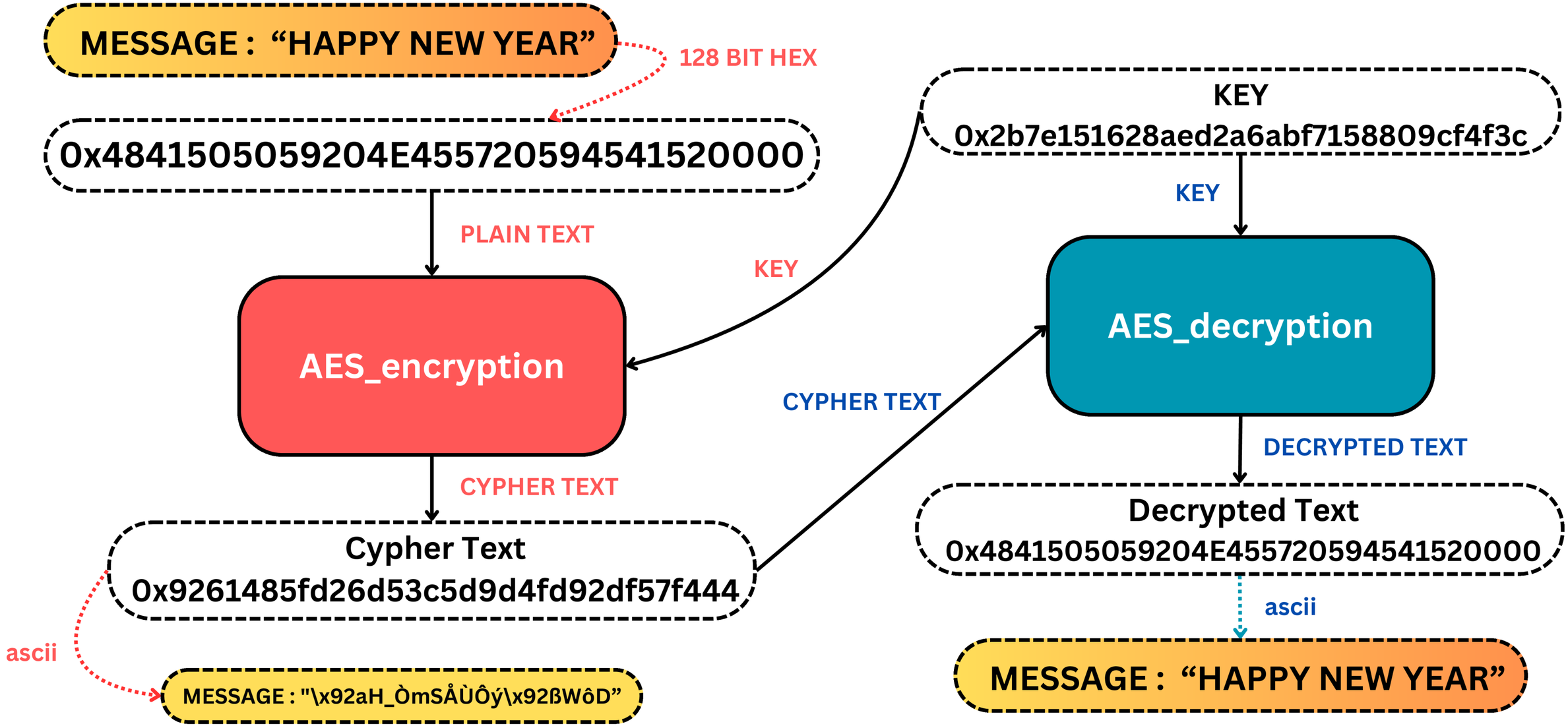


if the generated cypher_text for the
plain text = “0x00112233445566778899aabbccddeeff”
and
secret key =
“0x000102030405060708090a0b0c0d0e0f”
is
“0x69c4e0d86a7b0430d8cdb78070b4c55a”
then the Encryption is successfull

Testing Decryption



Testing Encryption and Decryption



References

- NIST AES
- AES - Wikipedia