IRS 2024

Advanced Encryption Standard

TE2002B.402: Design with programable logic

FINAL PRESENTATION FOR:

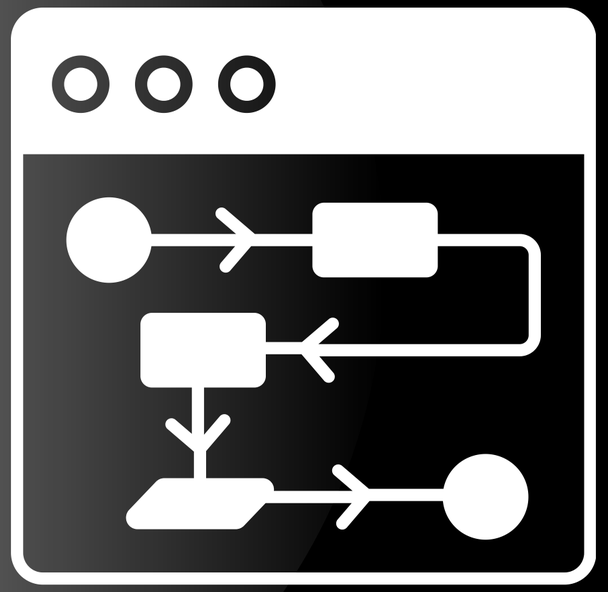
Professors & Intel

Agenda

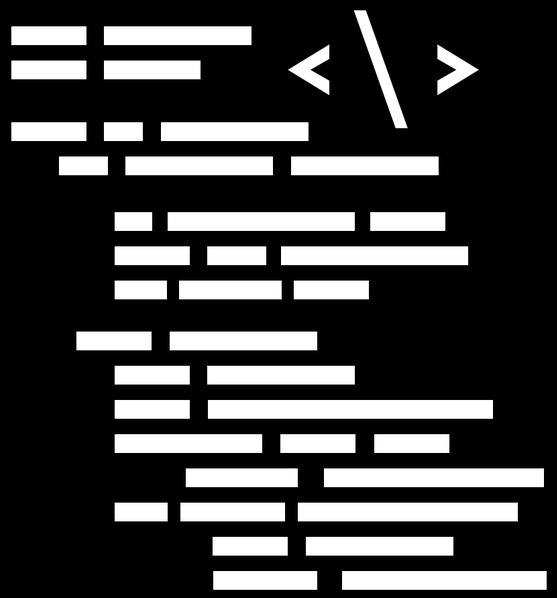
1. Project Overview
2. AES Algorithm
3. FPGA & VHDL
4. Components
    a. FSM
    b. Keys
    c. Encryption
    d. Decryption
5. Results

Project Overview

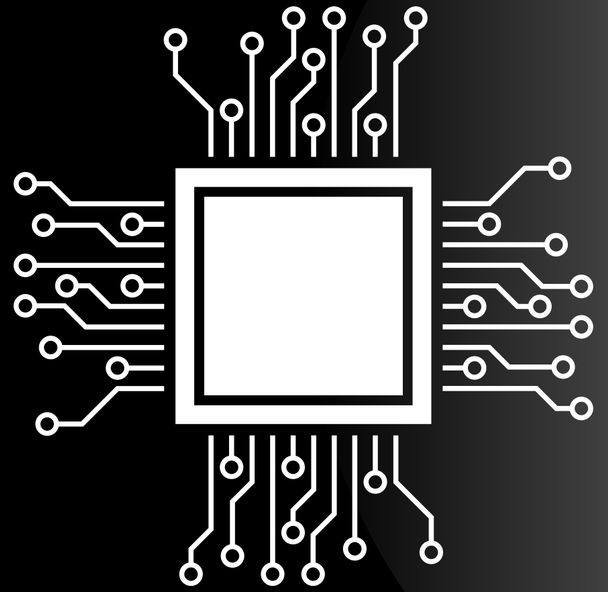
AES



VHDL

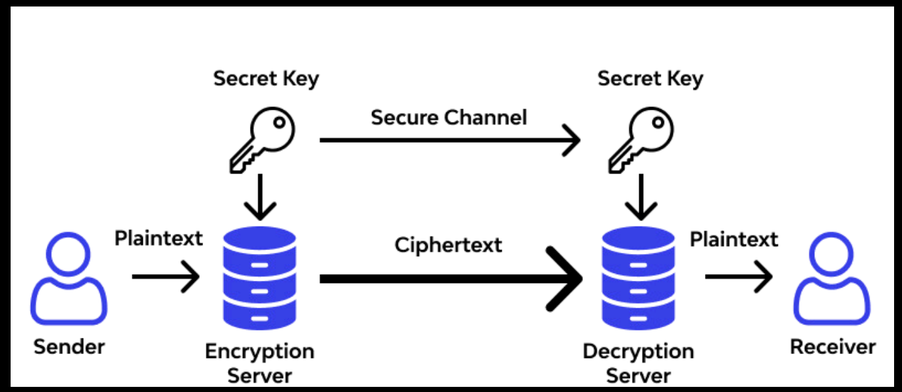


FPGA



AES Algorithm

Way of working

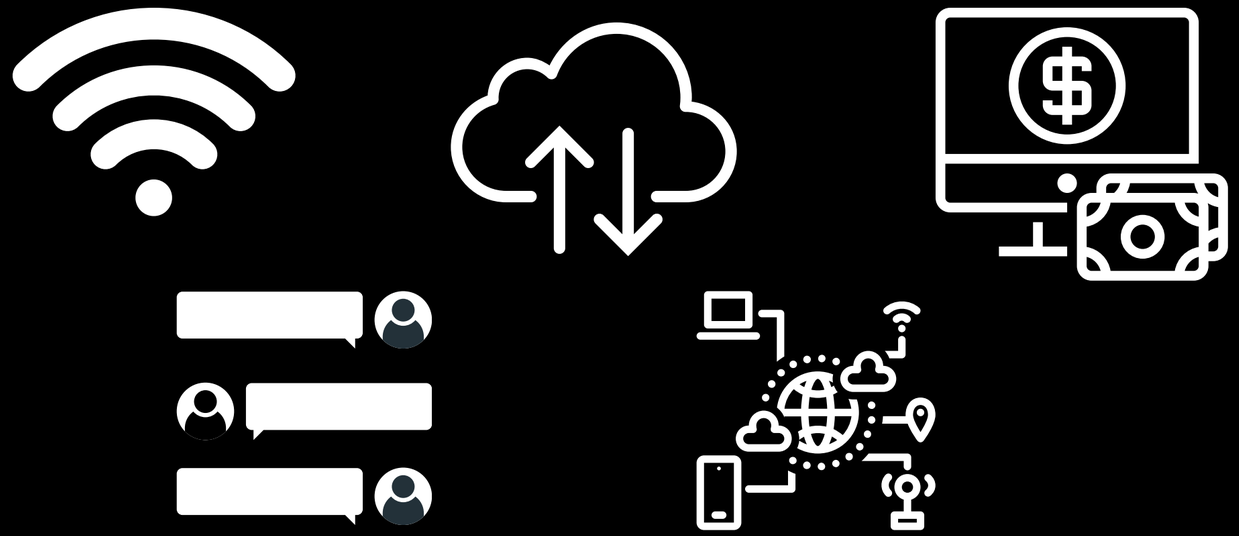


0	1	2	3
4	5	6	7
8	9	a	b
c	d	e	f

e	5	a	8
d	1	c	f
3	9	2	b
6	4	0	7

5	a	8	e
1	c	f	d
9	2	b	3
4	0	7	6

Use of AES



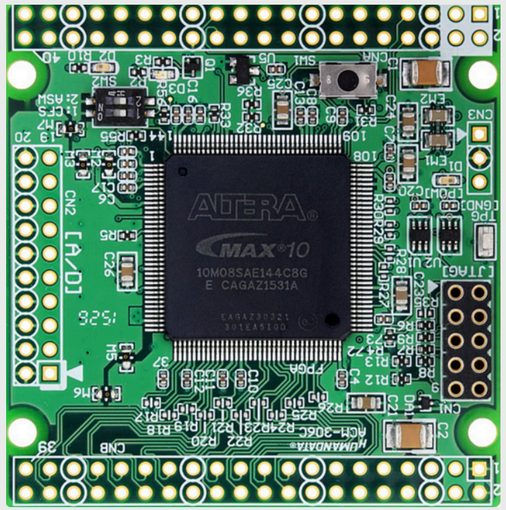
FPGA & VHDL

FPGA

- Field Programmable Gate Array
- Integrated Circuit to implement custom Designs
- Used for specific Tasks (Algorithms) with enhanced performance and limited use of resources
- Altera MAX 10

VHDL

- Very High Speed Integrated Circuit Hardware Description Language
- A standard in the World of HDL
- used to programm FPGA’s
- suitable for Simulation & Synthesis

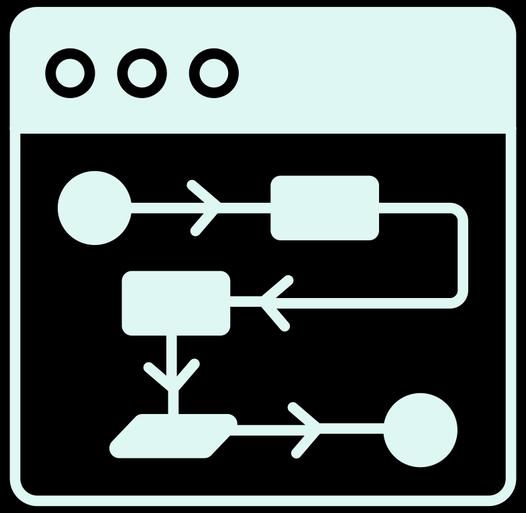


AES / Rijndael Algorithm

How it works

Plaintext

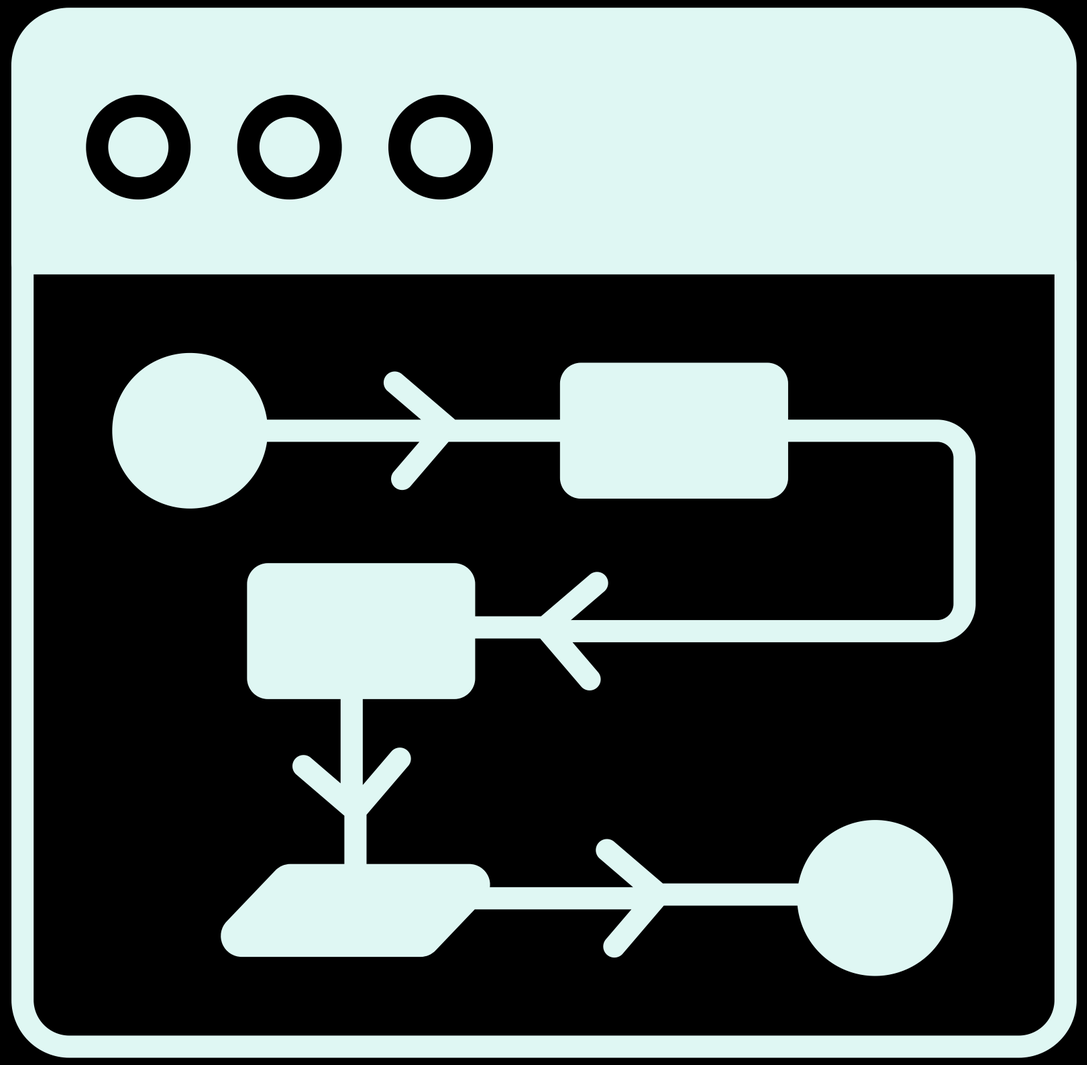
Encrypter



Decrypter

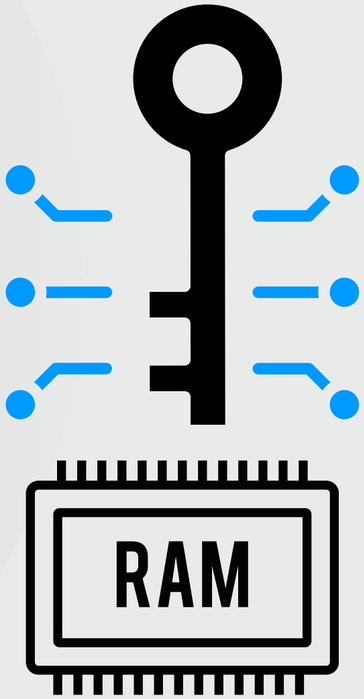
Ciphertext

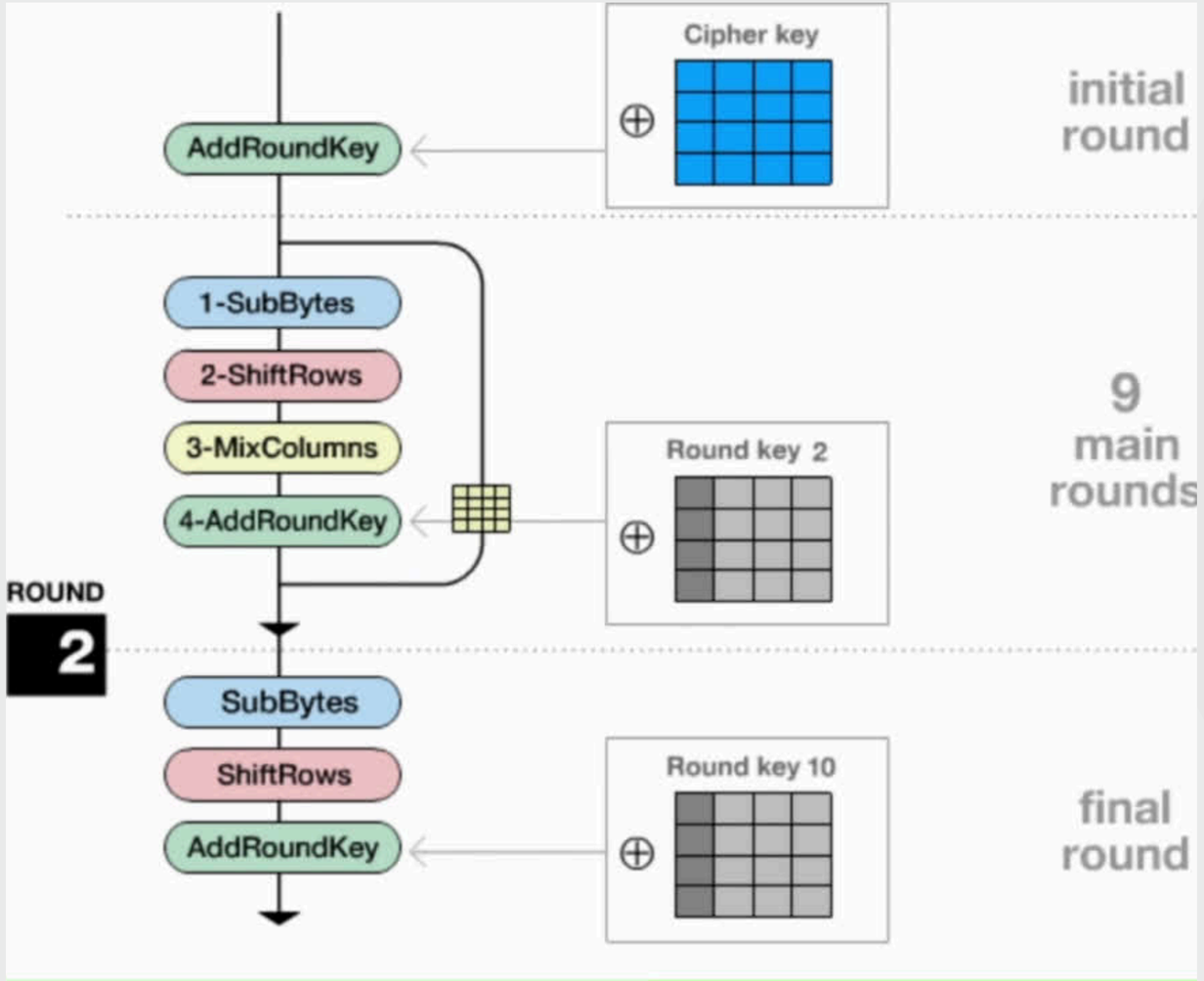
Encrypter



Encrypter

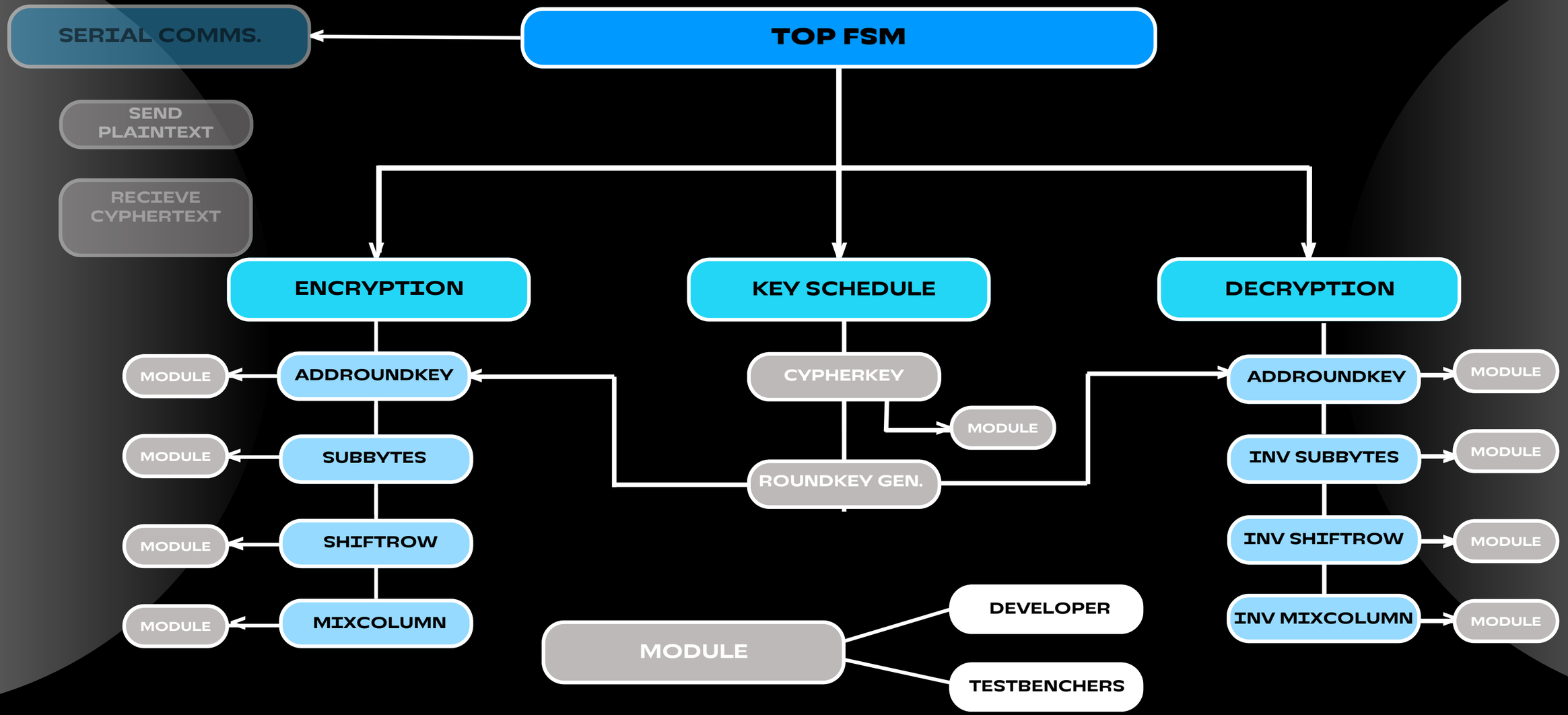
Key Schedule





Rijndael Cipher Animation By Enrique Zabala

Project Structure

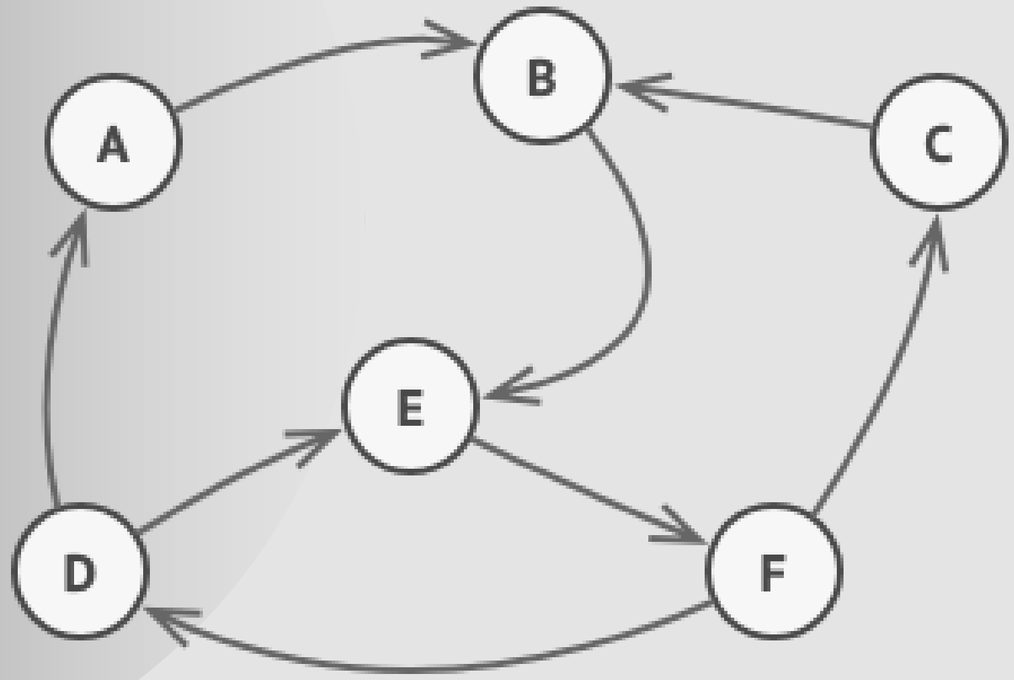


Top Level & FSM

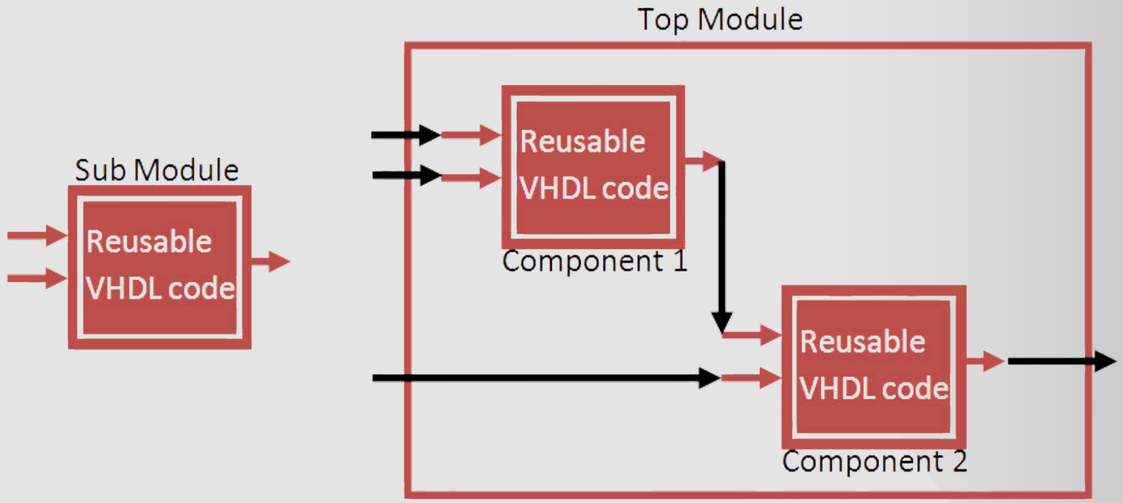
Highest level of the project structure hierarchy, where the finite state machine, components and their respective parameters are defined.

Importance of Top-Level Design

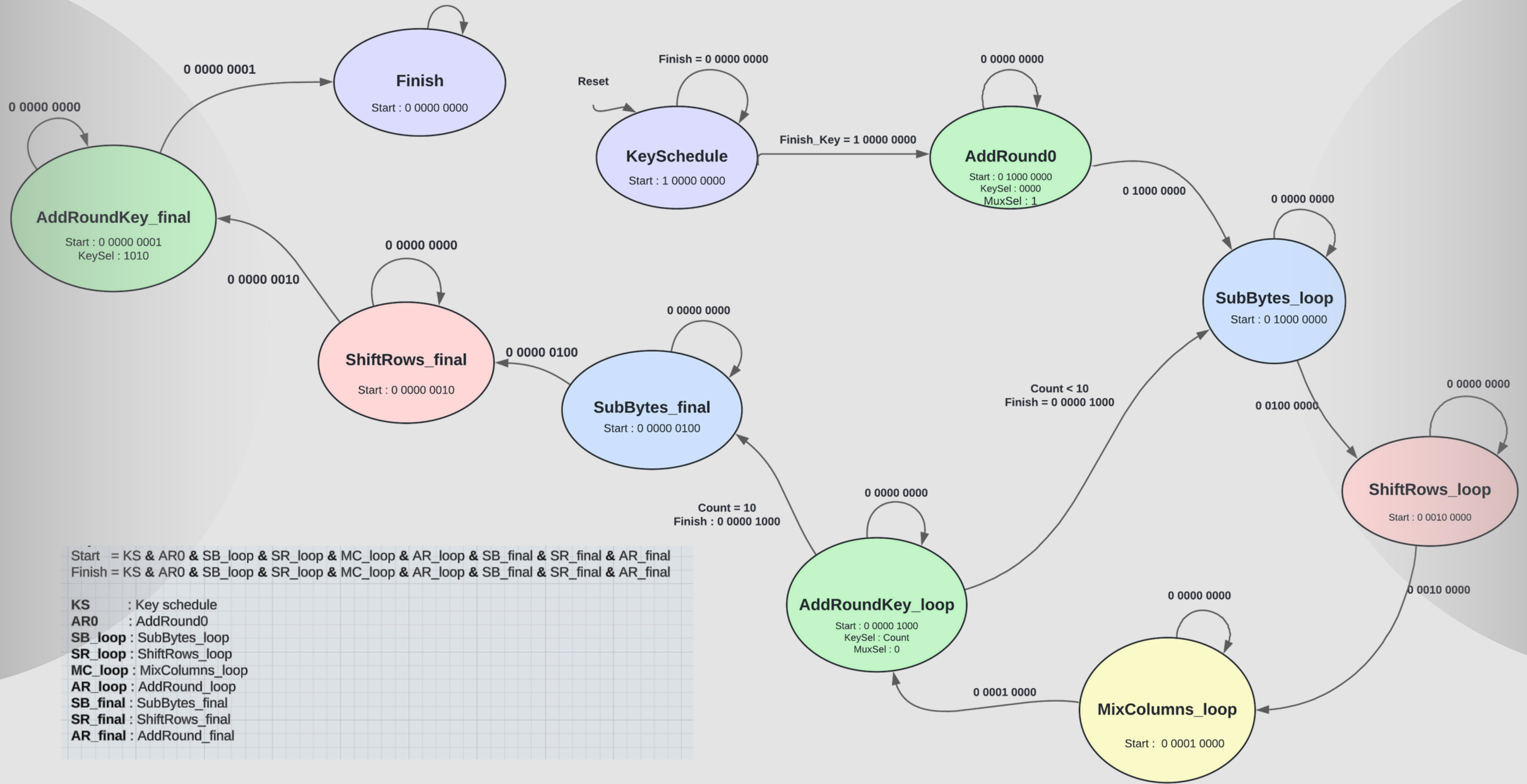
Finite State Machine



Component-Based Architecture



FSM



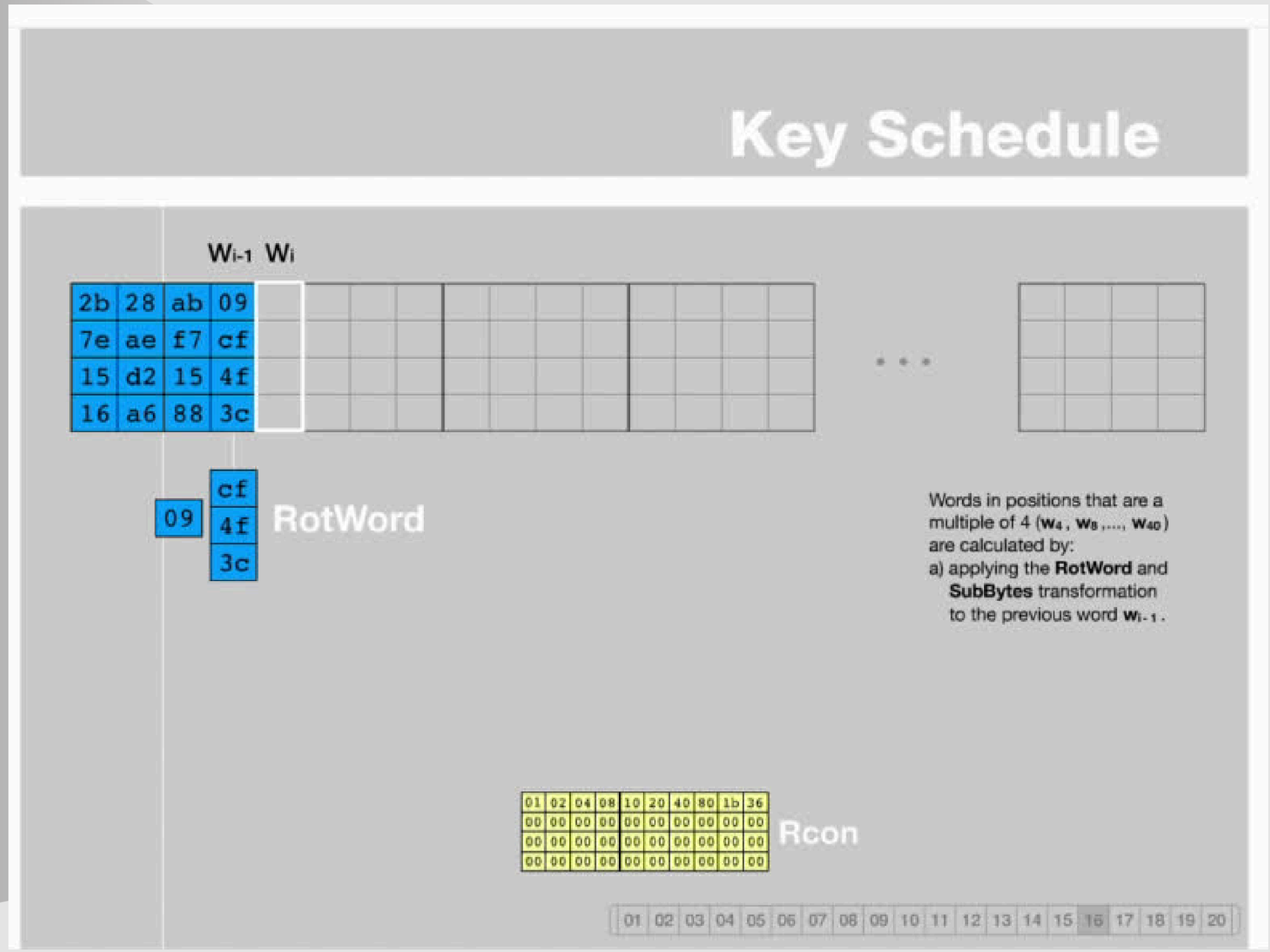
Start = KS & AR0 & SB_loop & SR_loop & MC_loop & AR_loop & SB_final & SR_final & AR_final
Finish = KS & AR0 & SB_loop & SR_loop & MC_loop & AR_loop & SB_final & SR_final & AR_final

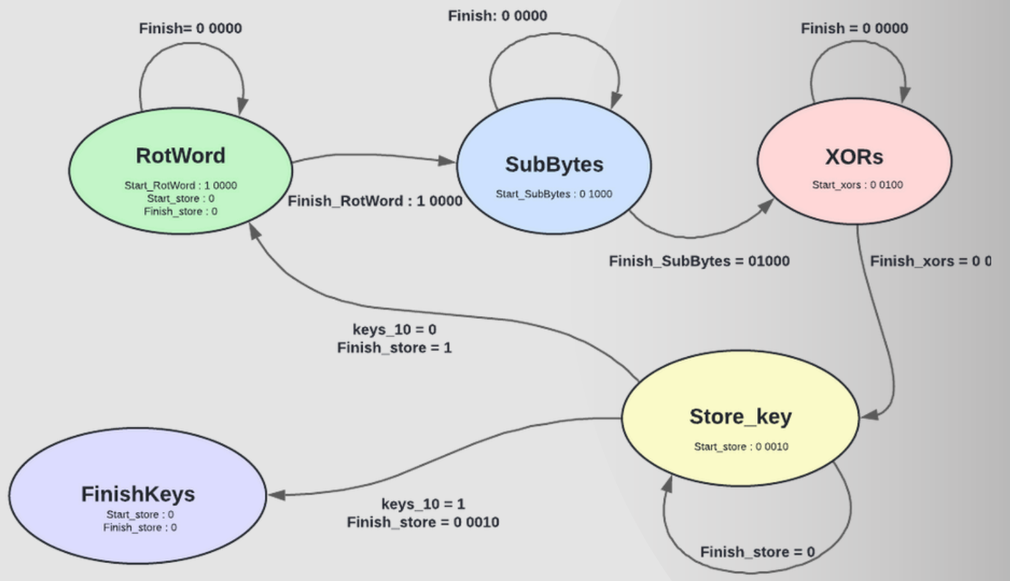
KS : Key schedule
AR0 : AddRound0
SB_loop : SubBytes_loop
SR_loop : ShiftRows_loop
MC_loop : MixColumns_loop
AR_loop : AddRound_loop
SB_final : SubBytes_final
SR_final : ShiftRows_final
AR_final : AddRound_final

Key Schedule

Generates the number of round keys specified by AES-128 from a source key. These round keys are later used by the encryption and decryption components.

Key Schedule

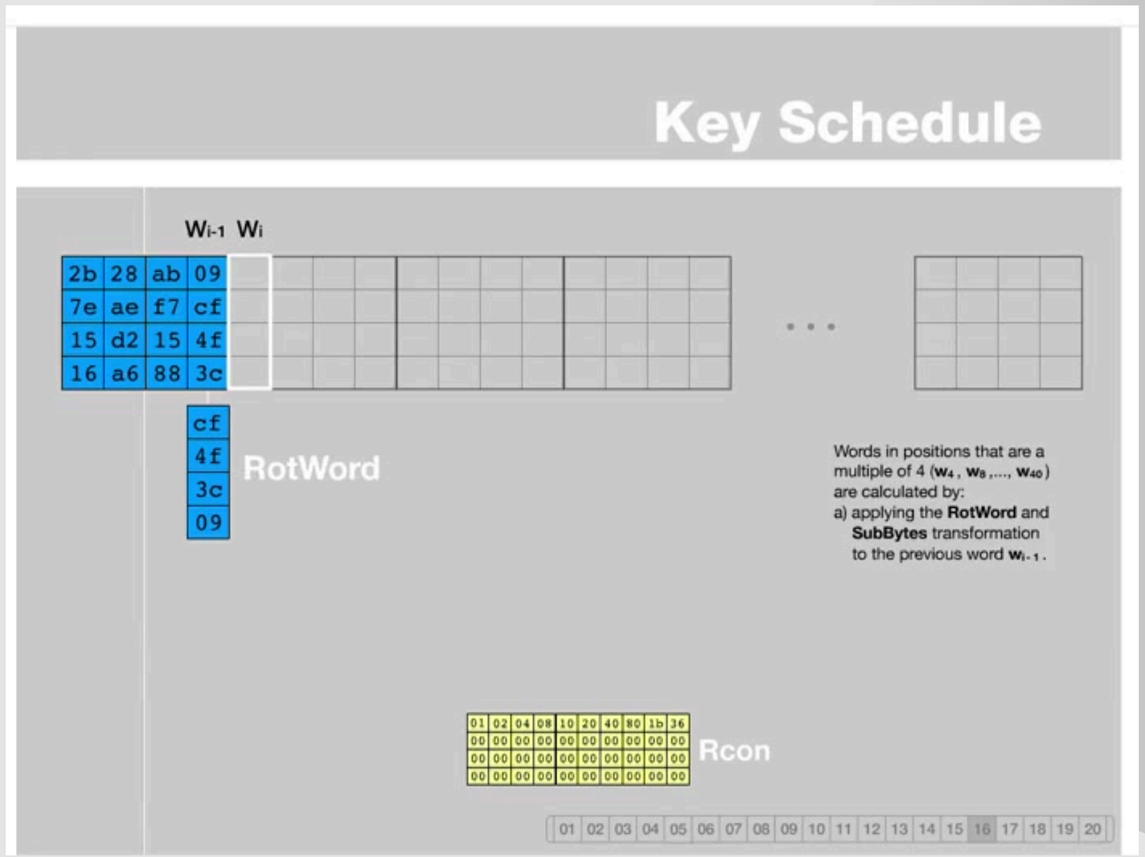




Module in Depth: SubBytes

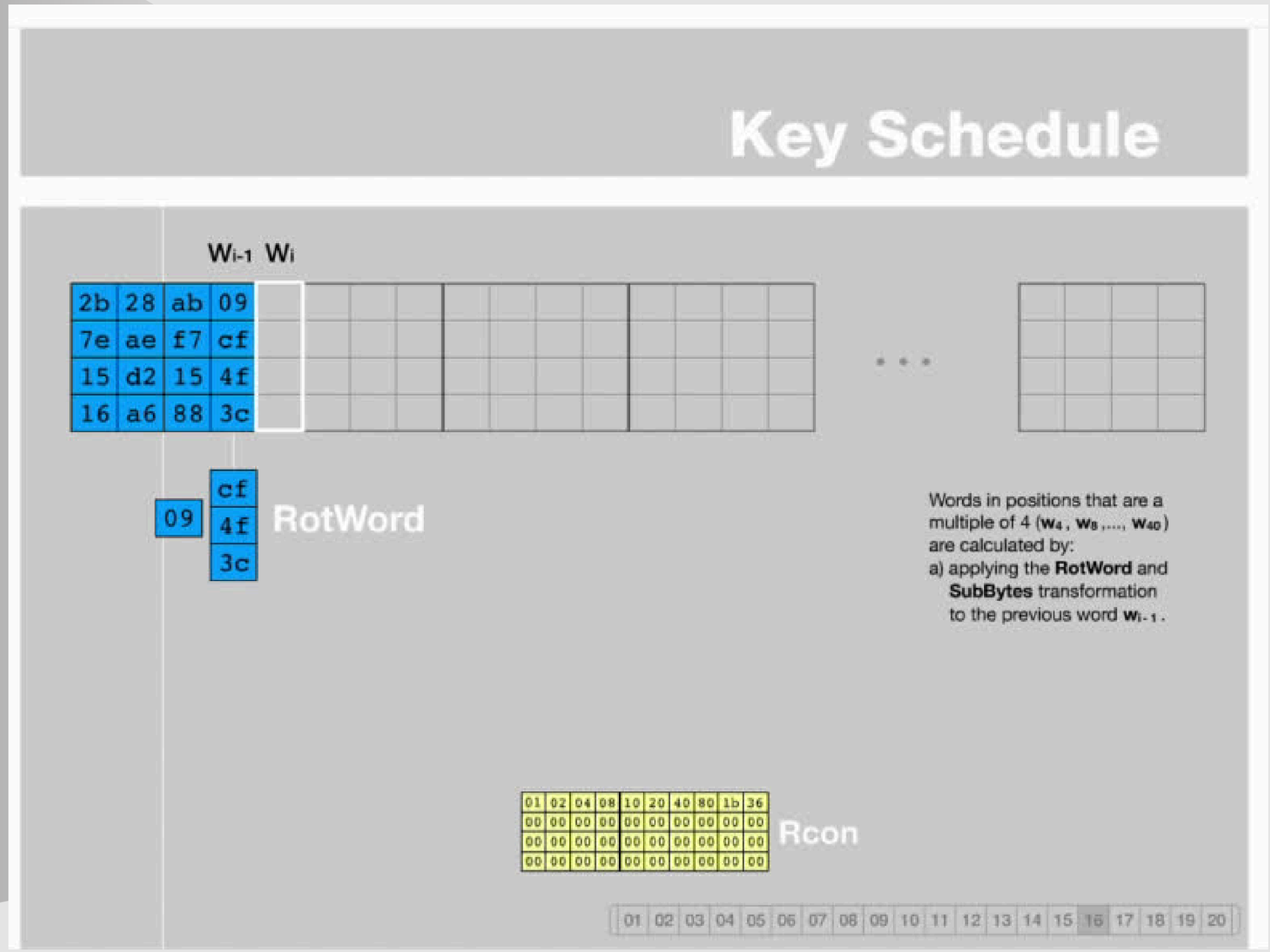
The SubBytes module performs a non-linear byte substitution on each byte of the received column of 4 bytes or 32 bits, provided by the previous module using an already defined S-box [substitution box].

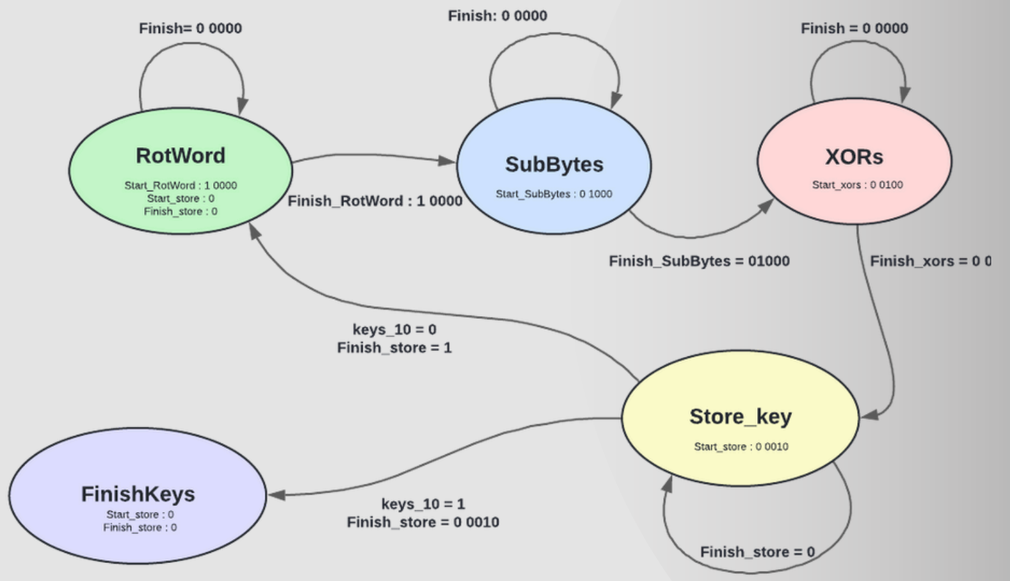
		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
X	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	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



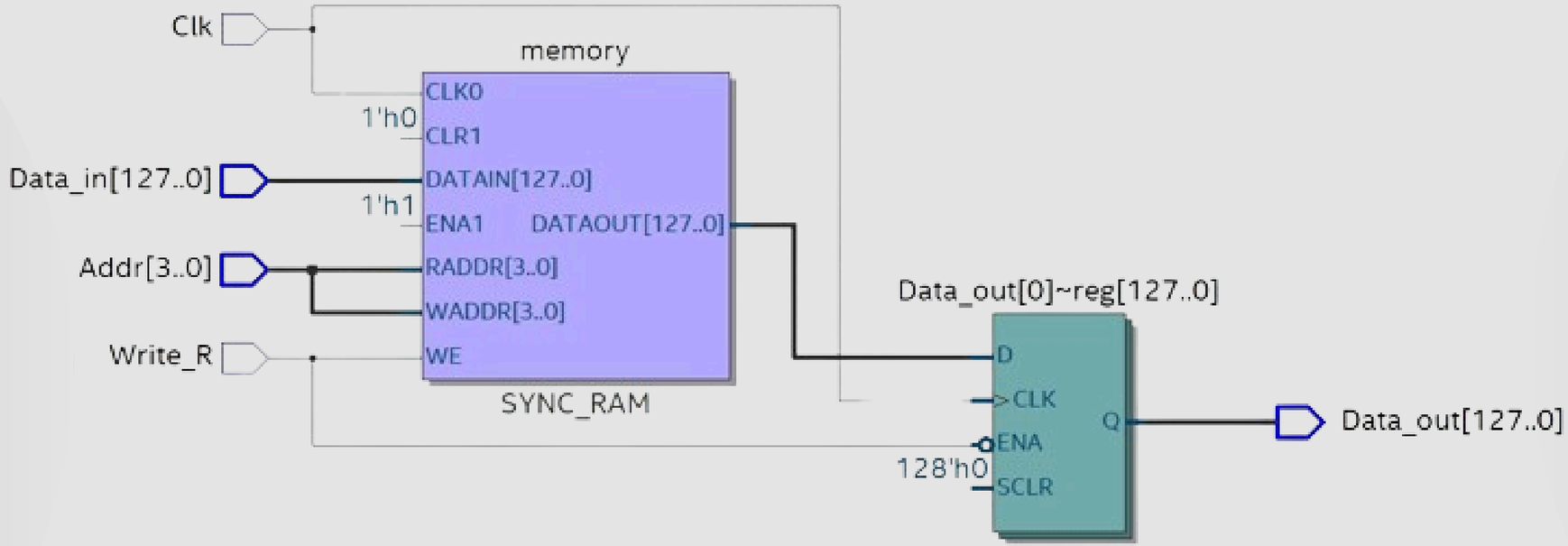
Rijndael Cipher Animation By Enrique Zabala

Key Schedule





Tech detail RAM



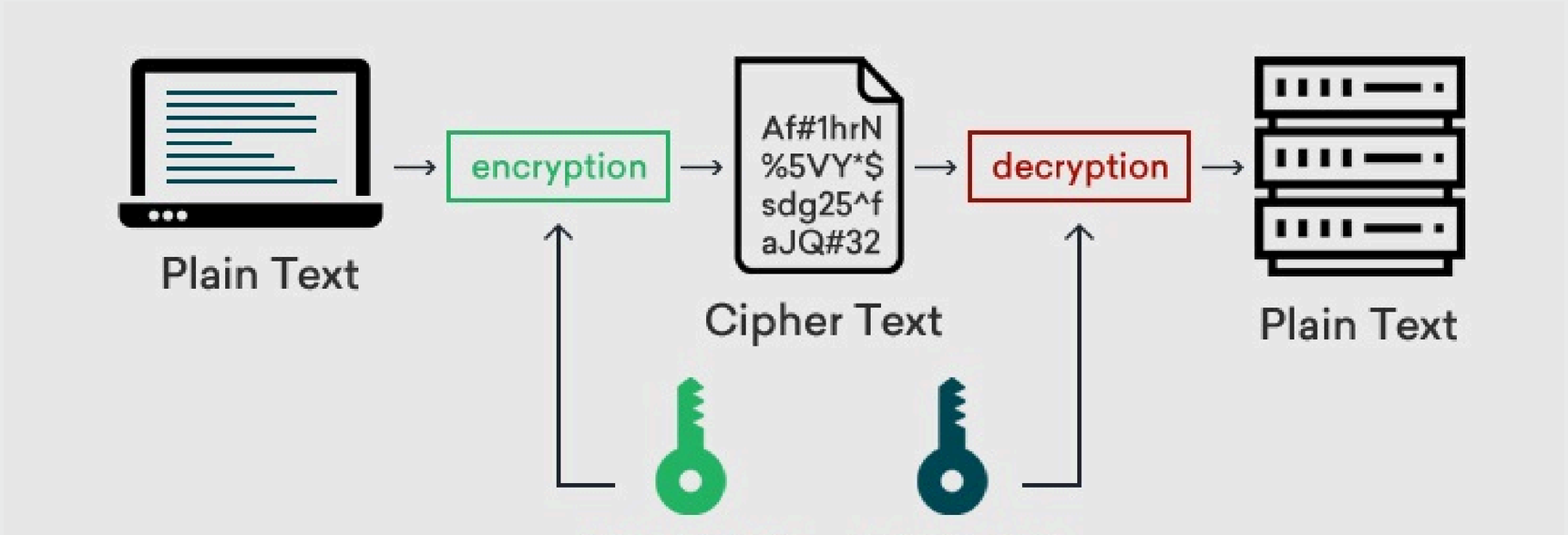
The RAM memory will be responsible for storing the 10 Round Keys. In order that the following modules can use the keys in the encryption and decryption processes

To access to each key, the user can enter a particular addres that will target a particular key in the RAM memory

Encryption

What the Encryptor Module does?

- tranforms a text into a ciphertext
- ensures confidetiality during transmition or storage



the submodules

AddRoundKey

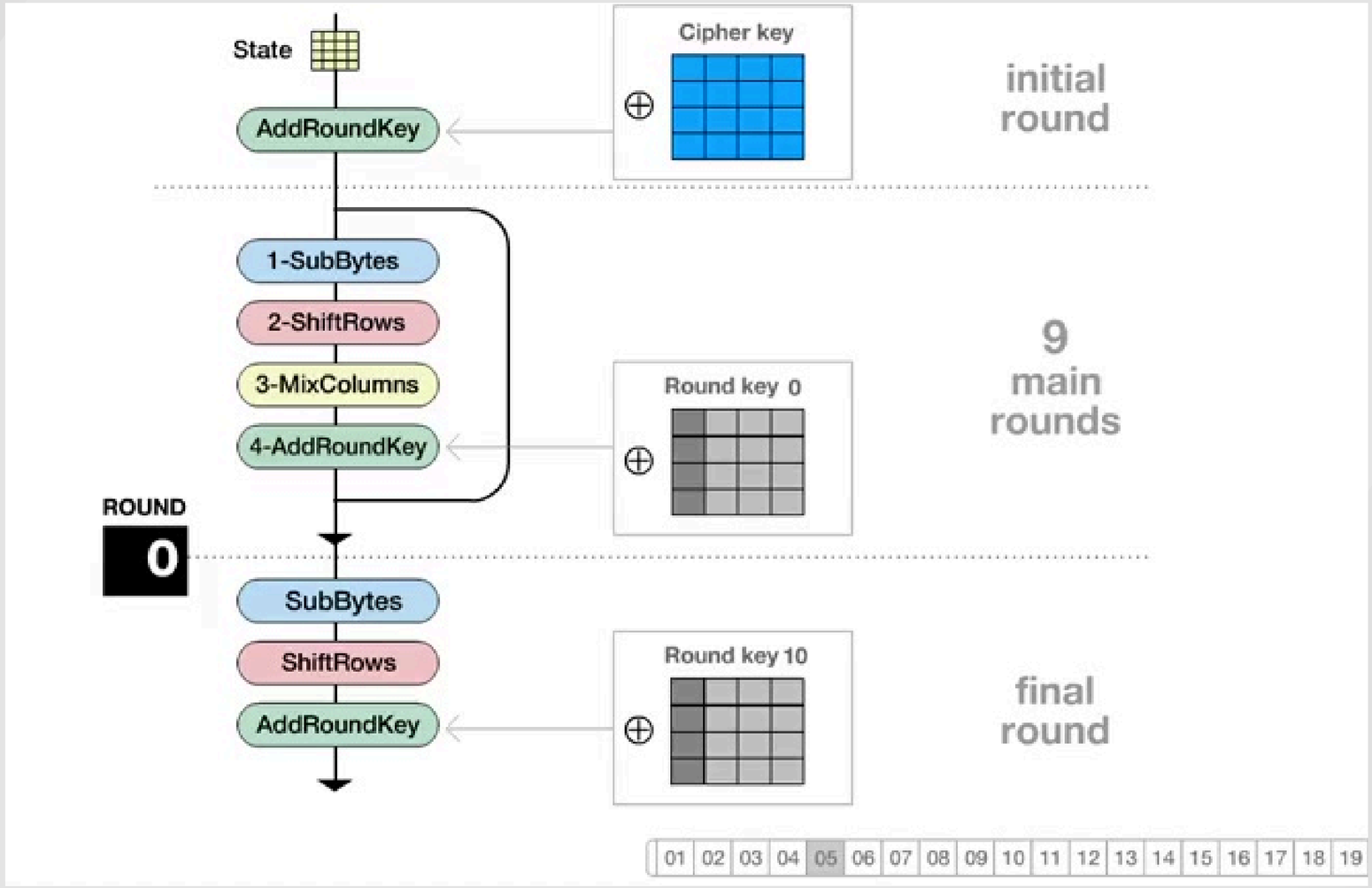
SubBytes

ShiftRows

MixColumns

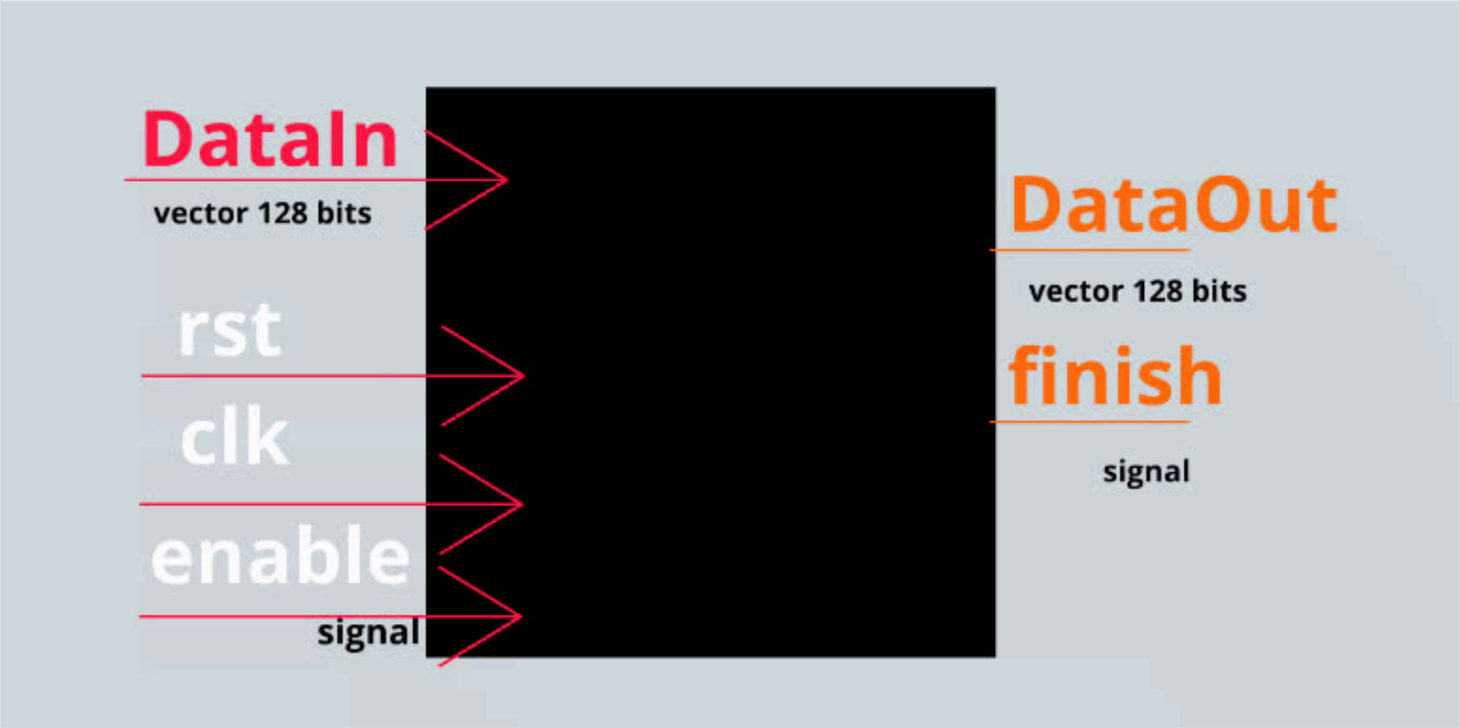
how the module operates?

serial process and always follows the same pipeline

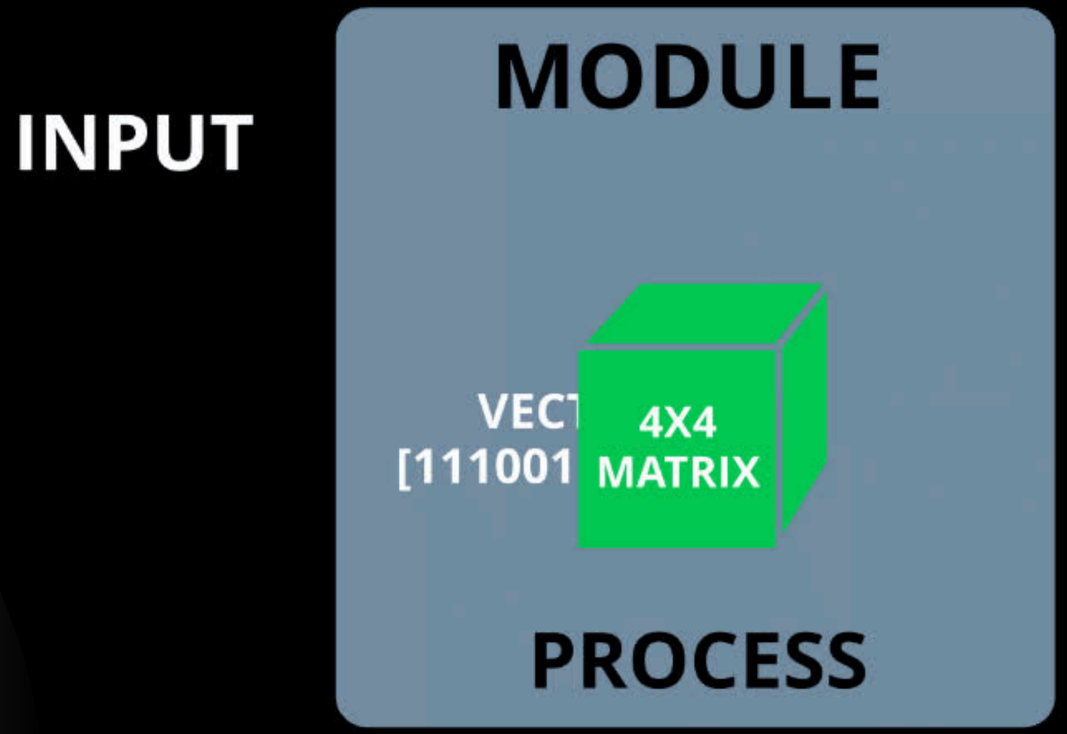


Rijndael Cipher Animation By Enrique Zabala

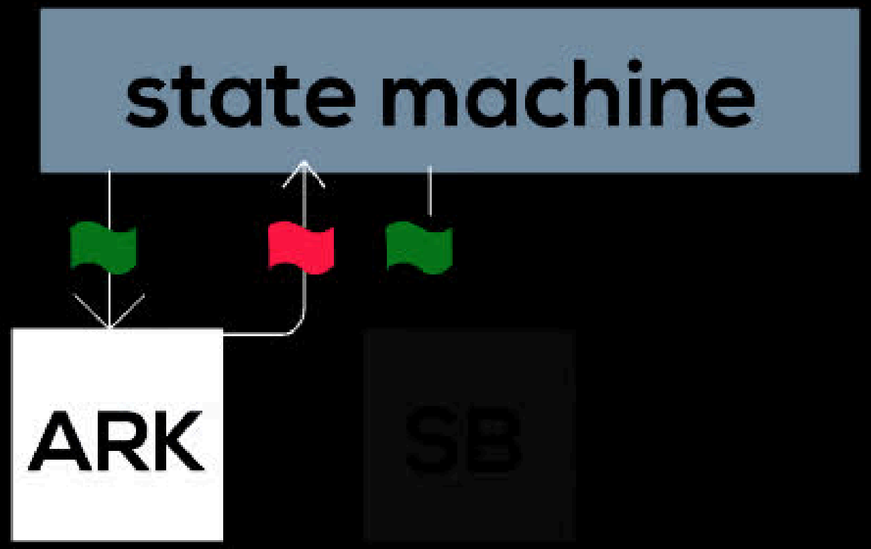
how we handle information?



how we handle information?



The process



Each sub-module has its own state. Once one has completed a task, it send a flag to the ESM, to start the next one.

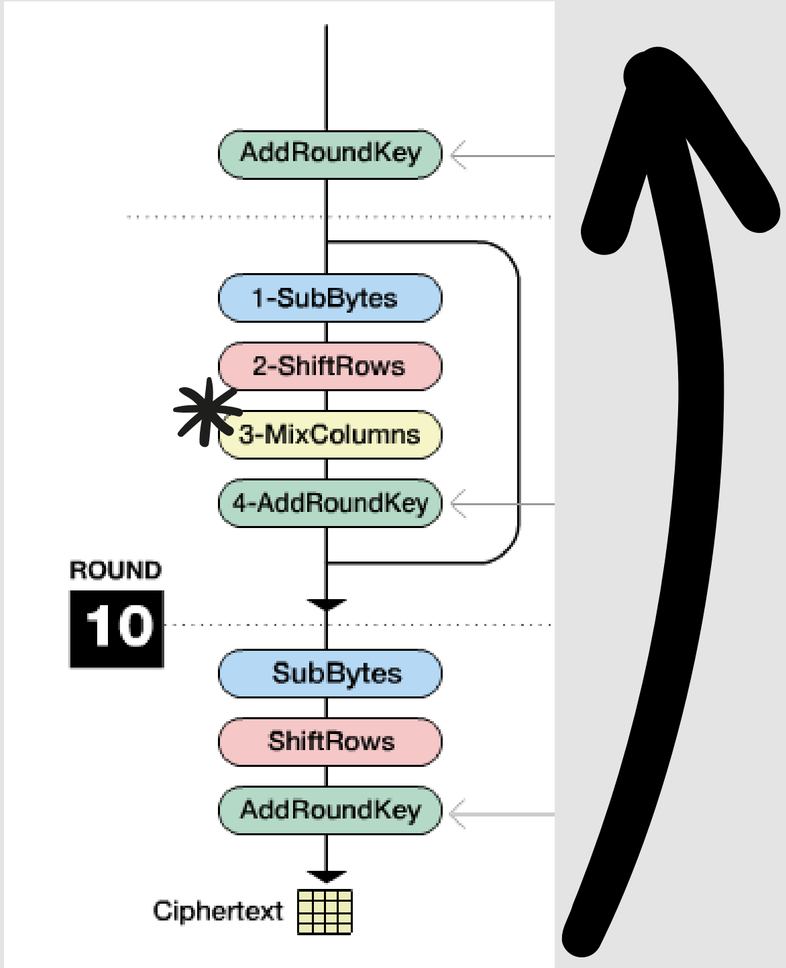
why this helps?
organized, undestandable, easy to fix or modify

Decryption

Inverse task of the encryption component, symmetrically composed of 4 inverse components. It handles the task of decrypting the ciphertext produced by the encryption process.

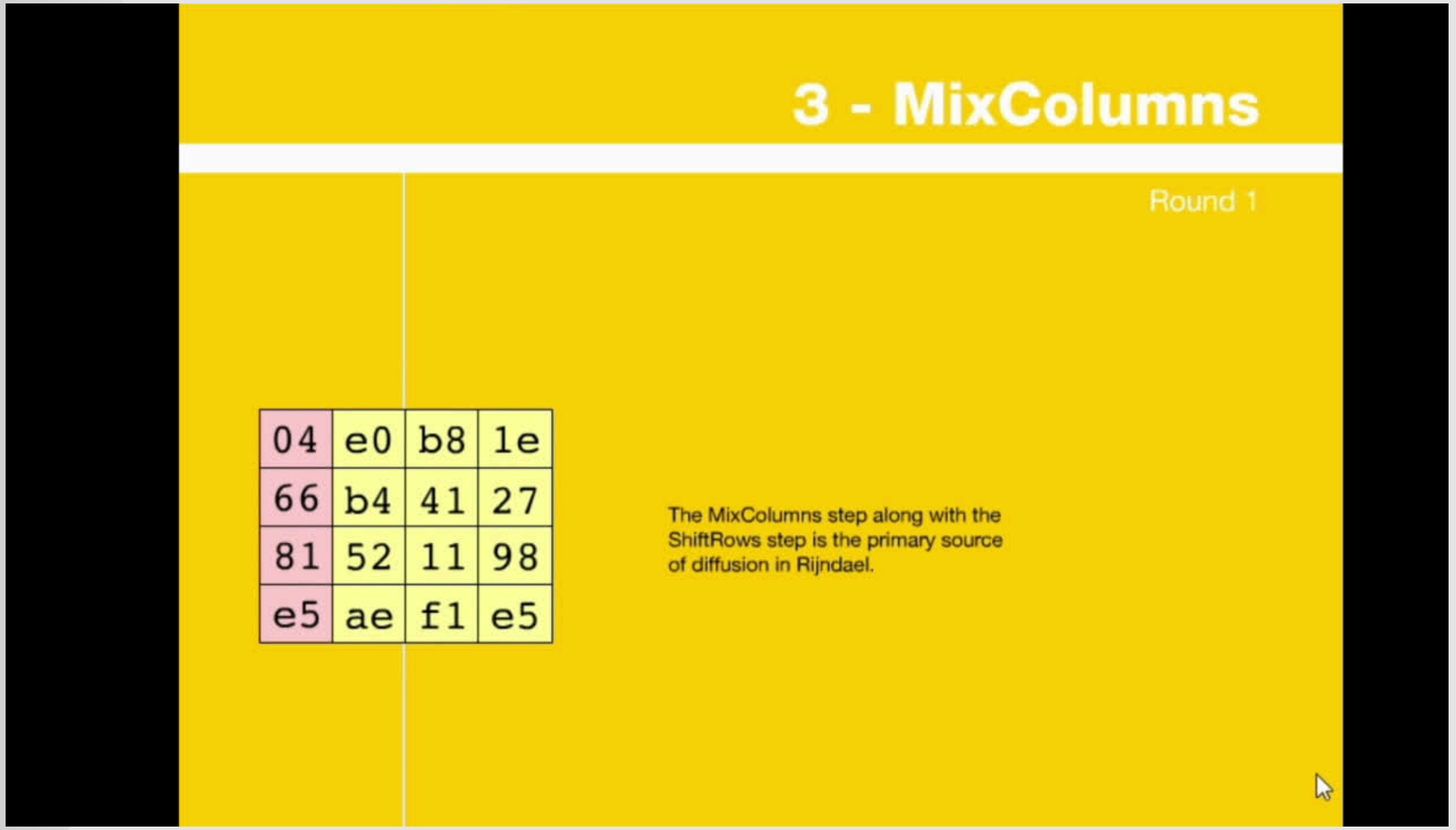
Decryption

Inverse Cipher process



MixColumns

Module

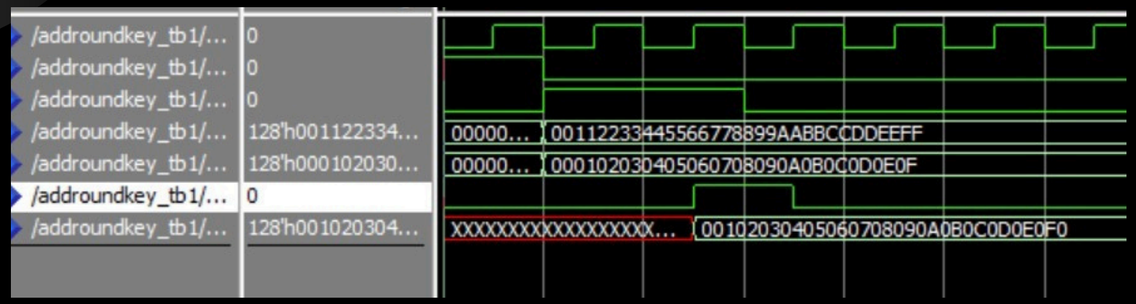


Rijndael Cipher Animation By Enrique Zabala

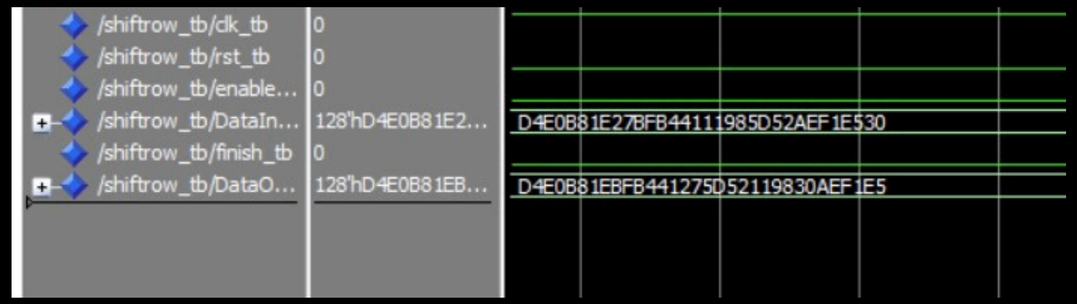
- ***Complexity***
- ***Implementation***
- ***Galois Fields***
- ***Importance***

Results

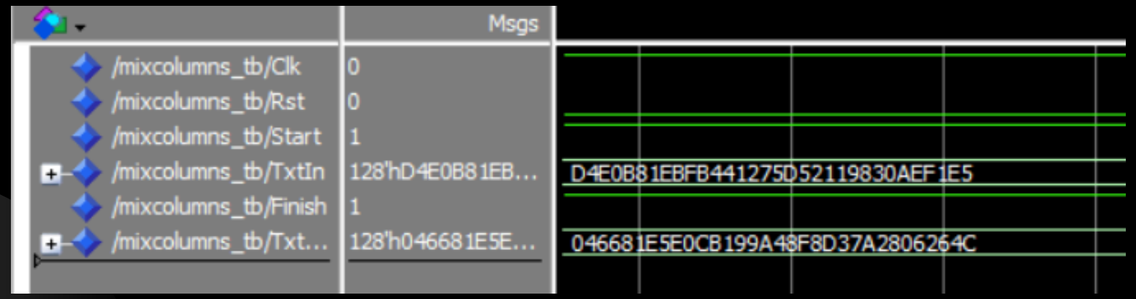
Encryption modules



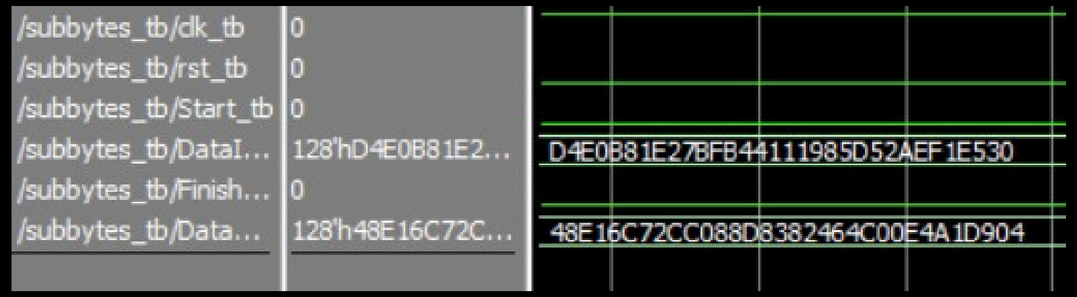
AddRoundKey



ShiftRows



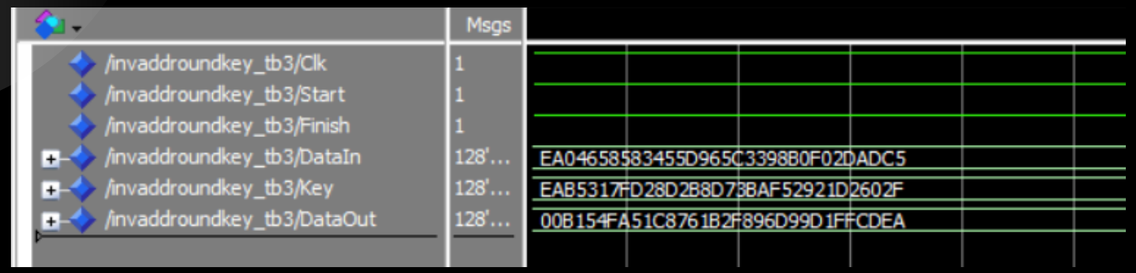
MixColumns



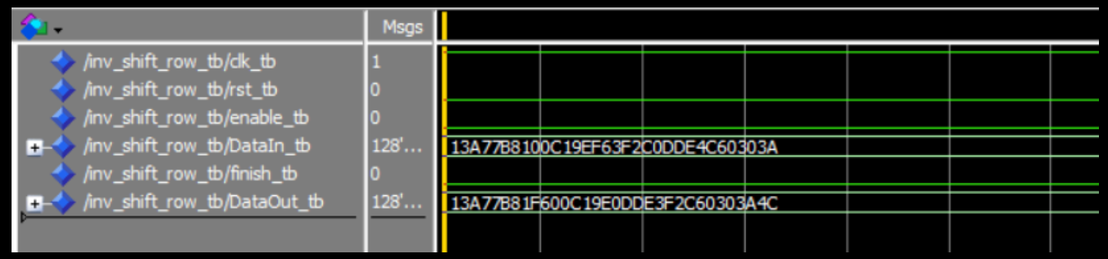
SubBytes

Results

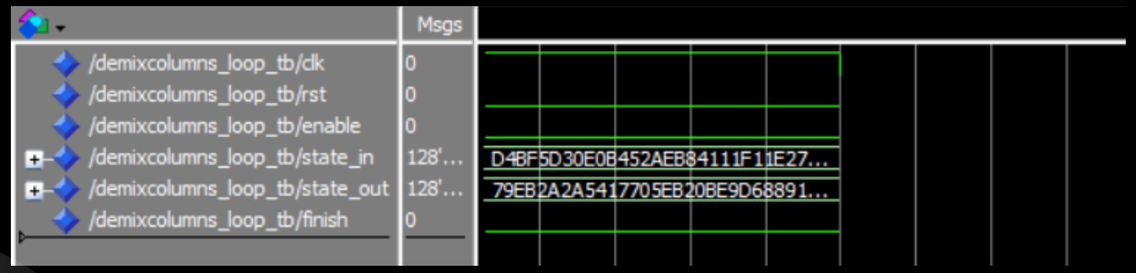
Decrypter modules



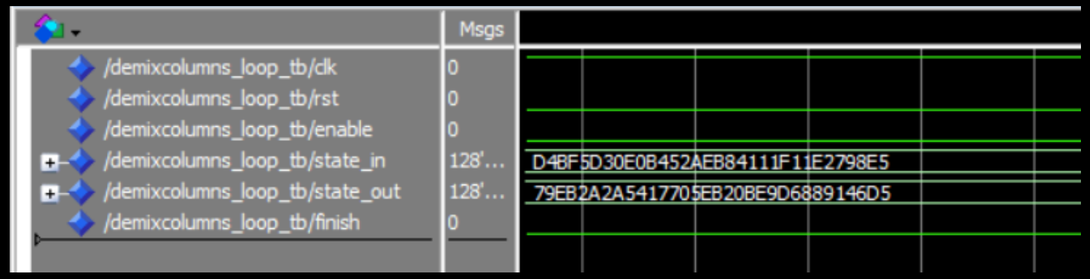
AddRoundKey



ShiftRows



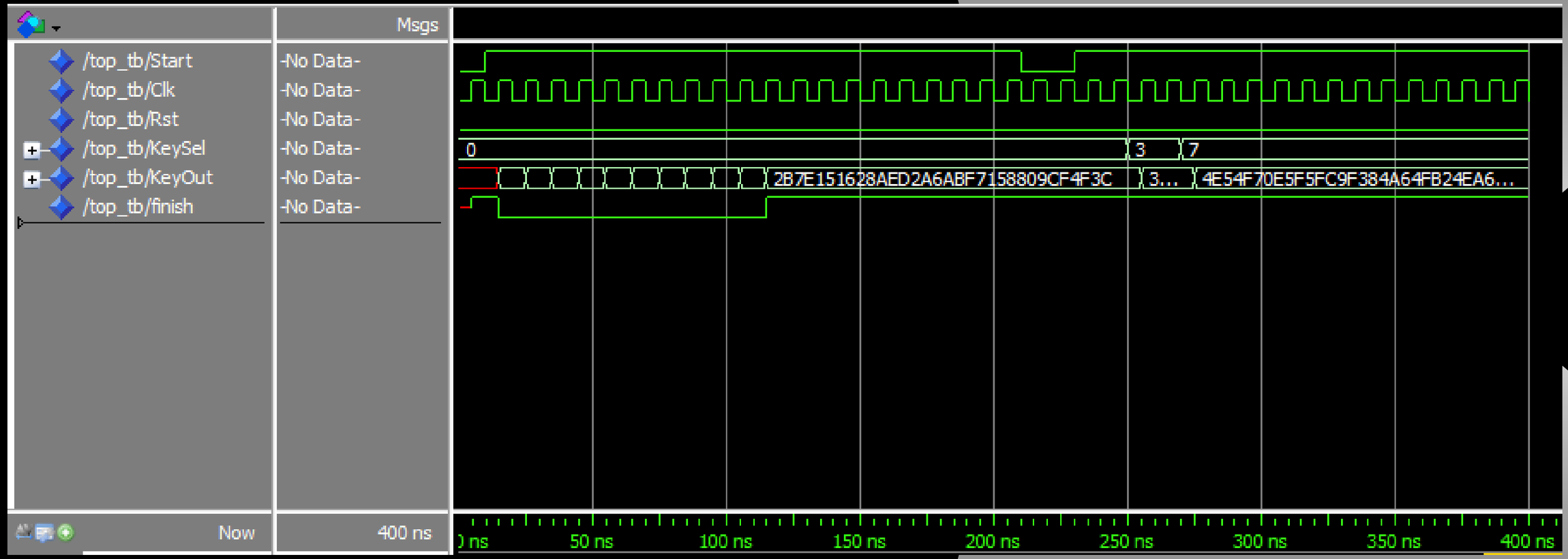
MixColumns



SubBytes

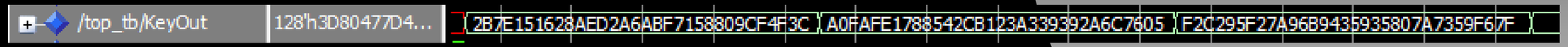
Up next...

Key Schedule
full implementation

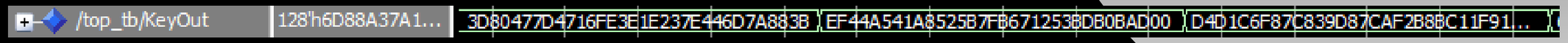


Key Schedule

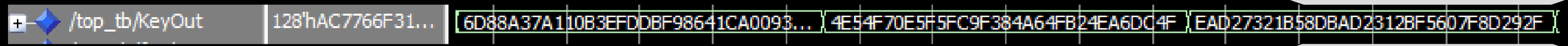
Cypher key & Keys 0-1



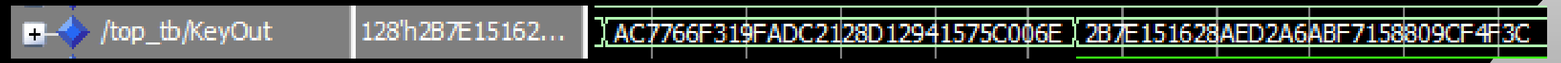
Keys 2-4



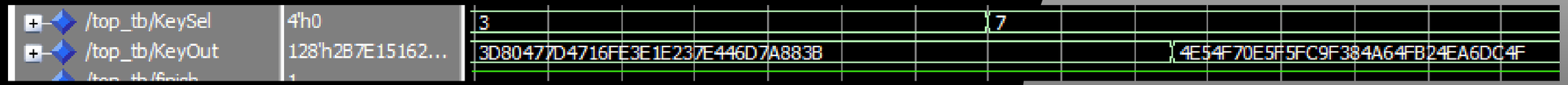
Keys 5-7



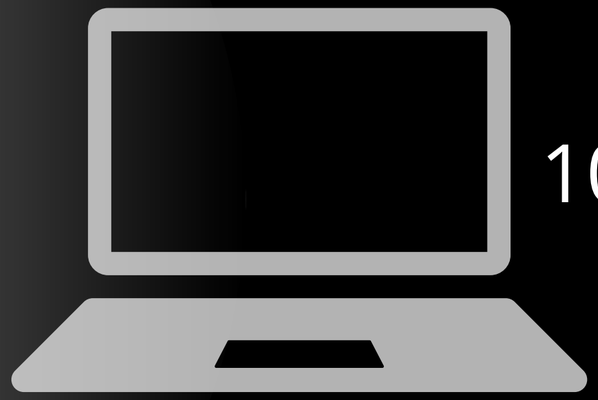
Keys 8-9



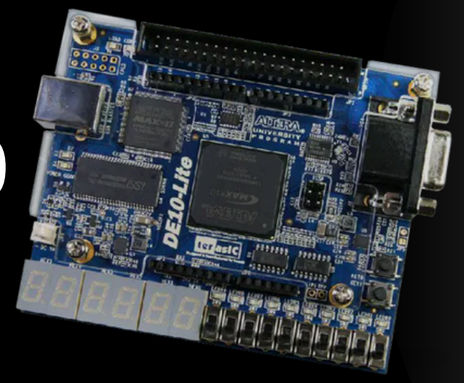
Key Selector



Serial Comunicator



10101101010101100010100



Thank You

we appreciate your time and attention.

Q&A

Collaborators

Top Level

Component Lead
Alexa Gonzalez
Helena Molina Jiménez

Developers
Emanuel Josue Vega Gonzalez
Ricard Esteve Catalá Garfias
Guadalupe Paulina López Cuevas

Key Schedule

Component Lead
Israel Sandoval Sánchez
Jose Alberto Aguilar

Developers
Sebastián Castellanos Rodríguez
Emanuel Josue Vega Gonzalez
Mario Godínez Chavero
Kevin Alejandro Ramirez Luna
José Eduardo Viveros Escamilla
Grant Nathaniel Keegan
Diego Quezada Colorado
Jonathan Arles Guevara Molina

Encryption

Component Lead
Karina Maldonado Murillo
Diego Gerardo Sánchez Moreno

Developers
Helena Molina Jiménez
Karla Zoé Rebolledo Argueta
Héctor Gúmaro Guzmán Reyes
Arturo Lopez Garcia
Tomás Pérez Vera
Ulises Carrizalez Lerín
Daniel De Regules Gamboa
Emanuel Josue Vega Gonzalez
Ricard Esteve Catalá Garfias
Paul Park

Decryption

Component Lead
Fiona Stasi
Jose Angel Huerta Rios

Developers
Alexa Jimena González Lucio
Brisa Itzel Reyes Castro
Santiago Reynaldo Aguilar Vega
Edgar Roann Santillan Bernal
Roberto Carlos Pedraza Miranda
Xavier Alfonso Barrera Ruiz
Alejandro Araiza Escamilla
Luis Adrián Uribe Cruz

Administrative

General Coordinator Aldrick Victor Tadeo Arellano
Supervisor Karina Maldonado Murillo
Version Control Admin Ricardo Sierra Roa, Ezzat Alzahouri Campos
Presentation Manager Beda Zünd

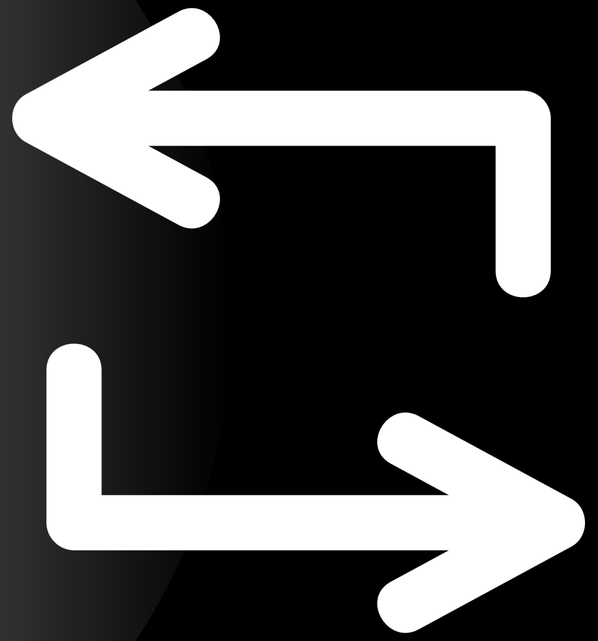
Professors

Pedro Nájera García
Rick Swenson Durie

Tecnológico de Monterrey

@TecdeMonterrey

Shift Rows Module



The nature of this module is just the exchange of bits within the vector, treating this as a matrix of four rows by four columns, where each of the cells of this contains 8 bits of this, whose arrangement goes from the most significant bit to the least significant, in an order from left to right and from top to bottom. This exchange will be carried out based on the displayed cells of the matrix, whose position change will be made from left to right in an iterative ascending way per row, being that for the first row no iteration will be made; one for the second row; two for the third row; and three for the fourth row.

Mix Columns Module

For the implementation of Mix Columns Module the key components were the Galois Finite Fields and the 128-bit Input Vector.

For the correct data management and processing the vector was transformed into the 4x4 Input Matrix.

The data processing consists of multiplying each column of the 4x4 Input Matrix by the Galois Matrix, followed by modular additions with respect to an irreducible polynomial in GF[2^3].

The final step was to return and transform the 128 bits in the 4x4 Input Matrix in a 128-bit Output Vector.

SubBytes Module

The implementation of this module was carried out with the following theory:

1. **Inputs and Outputs**:
   - **TxtIn**: Input data to be processed, organized as a 128-bit vector.
   - **Start**: Signal to start the processing operation.
   - **Clk**: Clock signal for synchronous operation.
   - **Rst**: Reset signal.
   - **Finish**: Output signal indicating operation completion.
   - **TxtOut**: Output processed data, also organized as a 128-bit vector.
2. **Internal Structures:**
   - State: Represents a 4x4 array of bytes [state_array]. It holds the intermediate state during the byte substitution process.
   - SBox: Represents the substitution box used for byte substitution [SBox_array]. It maps each byte value to a different byte value according to a predefined table [SBox].
3. **Operation:**
   - Upon receiving a Start signal and a rising clock edge, the module begins processing.
   - It uses the input data [TxtIn] to populate the State array after performing byte substitution using the SBox.
   - The processed data is then outputted through TxtOut.
   - The Finish signal indicates the completion of the operation.
4. **Byte Substitution:**
   - Byte substitution involves replacing each byte of the input data with a corresponding byte from the SBox.
   - The SBox is a fixed lookup table where each input byte value is substituted with a specific output byte value based on its position in the table.
   - In the provided code, the State array is populated with byte substitutions based on chunks of the input data [TxtIn].