Camellia Encryption Algorithm

Table of Contents

[Introduction 2](#_Toc43914172)

[Camellia 2](#_Toc43914173)

[Key scheduling part 2](#_Toc43914174)

[Data Randomization part 3](#_Toc43914175)

[Camellia Encryption Algorithm Interface 5](#_Toc43914176)

[Signal Description 5](#_Toc43914177)

[Basic Transfer 6](#_Toc43914178)

[Hierarchy of Modules 6](#_Toc43914179)

[Control path 7](#_Toc43914180)

[Description: 7](#_Toc43914181)

[Control Signals 8](#_Toc43914182)

[Data path 8](#_Toc43914183)

[Function – F 9](#_Toc43914184)

[Function – FL and Function – FLINV 9](#_Toc43914185)

[Key Generation for Function- FL and FLINV 10](#_Toc43914186)

[Round Key logic 11](#_Toc43914187)

[Example Scenarios – KEY PHASE 12](#_Toc43914188)

[Scenario 1 12](#_Toc43914189)

[Scenario 2 13](#_Toc43914190)

[Scenario 3 14](#_Toc43914191)

[Scenario 4 15](#_Toc43914192)

[Waveforms depicting the KEY PHASE state 16](#_Toc43914193)

[Example Scenarios – DATA RANDOMIZATION 17](#_Toc43914194)

[Scenario 1 17](#_Toc43914195)

[Scenario 2 18](#_Toc43914196)

[Scenario 3 19](#_Toc43914197)

[Scenario 4 20](#_Toc43914198)

[Waveform depicting the DATA\_RANDOMIZATION state 21](#_Toc43914199)

[Number of cycles for key phase 21](#_Toc43914200)

[Number of cycles for data randomization 21](#_Toc43914201)

# Introduction

## Camellia

Camellia was jointly developed by Nippon Telegraph and Telephone Corporation and Mitsubishi Electric Corporation in 2000. Camellia specifies the 128-bit block size and 128-,192-, and 256-bit key sizes, the same interface as the Advanced Encryption Standard (AES). Camellia is characterized by its suitability for both software and hardware implementations as well as its high level of security. From a practical viewpoint, it is designed to enable flexibility in software and hardware implementations on 32-bit processors widely used over the Internet and many applications, 8-bit processors used in smart cards, cryptographic hardware, embedded systems, and so on

Camellia Encryption algorithm has two steps

1. Key scheduling part
2. Data randomisation part

## Key scheduling part

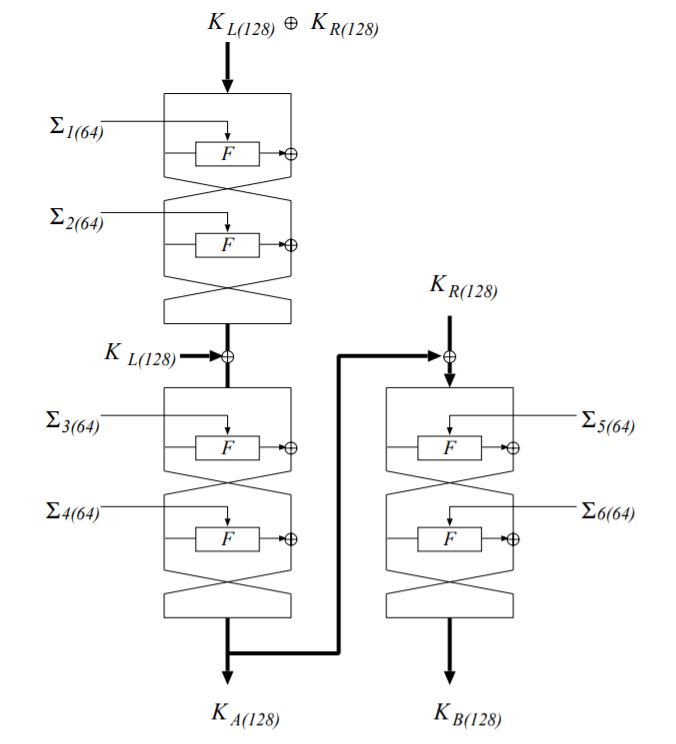


Figure 1: Key scheduling part

## Data Randomization part

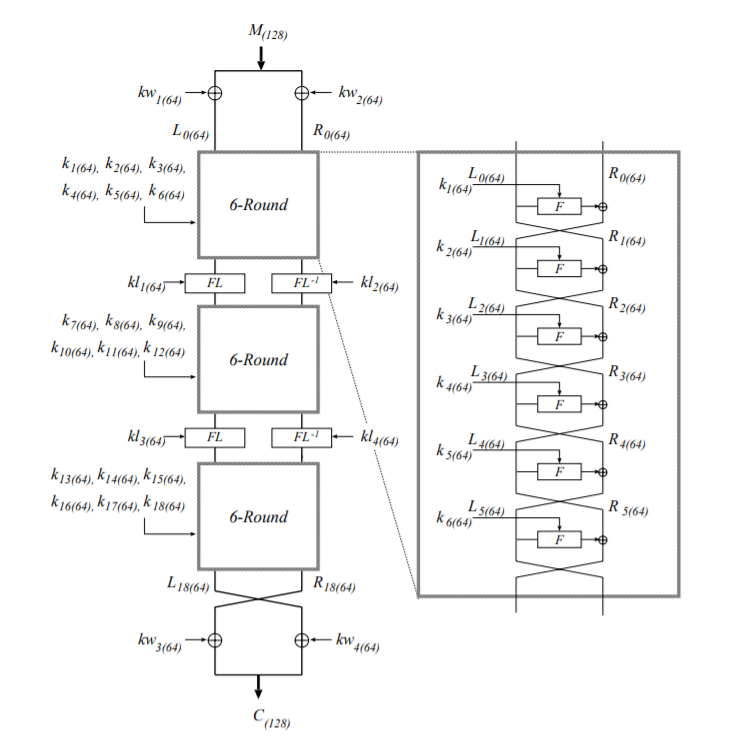


Figure 2: Encryption procedure for 128-bit key

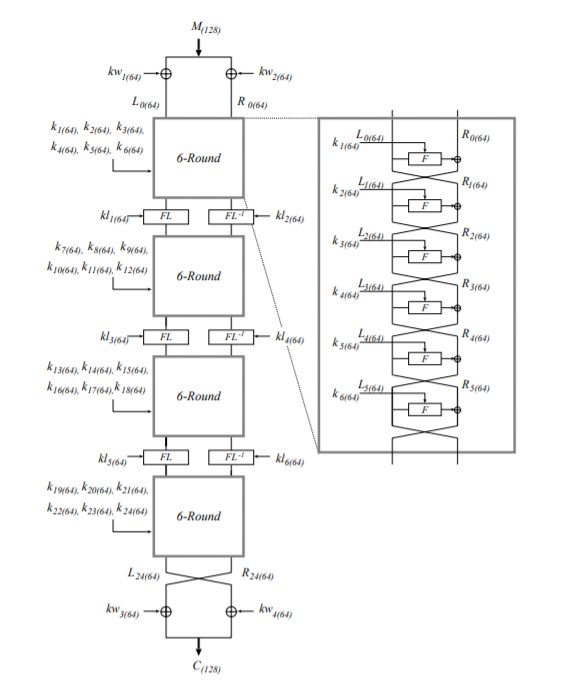


Figure 3: Encryption procedure for 192 and 256-bit key

# Camellia Encryption Algorithm Interface

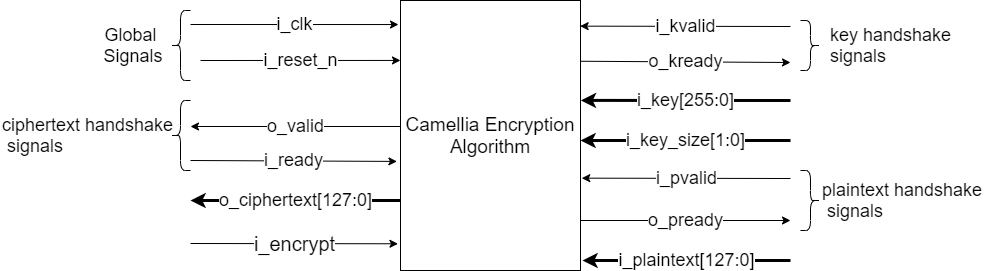


Figure 4: CAE interface

## Signal Description

|  |  |
| --- | --- |
| Name | Description |
| i\_clk | All signals are related to the rising edge of **i\_clk** |
| i\_reset\_n | The reset signal is active LOW and It resets the system. This is the only active LOW signal among all. |
| i\_kvalid | **i\_kvalid** is an input handshake signal when HIGH it says that the **i\_key** and **i\_key\_size** are valid inputs |
| o\_kready | **o\_kready** is an output handshake signal which when HIGH indicates that the module is ready to accept a new key and key size |
| i\_key | **i\_key** is **256** - bit encryption key |
| i\_key\_size | **i\_key\_size** is a **2- bit** number indicating the size of key for encryption  **00 – 128-bit key, 01 – 192-bit key, 10 – 256-bit key** |
| i\_pvalid | **i\_pvalid** is an input handshake signal when HIGH it says that the **i\_plaintext** is a valid input |
| o\_pready | **o\_pready** is an output handshake signal which when HIGH indicates that the module is ready to accept a new plaintext |
| i\_plaintext | **i\_plaintext** is **128**-bit block of data to be encrypted |
| i\_ready | **i\_ready** when HIGH indicates that the user has accepted the output ciphertext |
| o\_valid | **o\_valid** indicates that the cipher text output is valid |
| o\_ciphertext | **o\_ciphertext** is **128**-bit encrypted ciphertext of the given plaintext and key |
| i\_encrypt | **i\_encrypt** when HIGH indicates that the block of data has to be encrypted and when LOW indicates that the block of data should be decrypted. |

## Basic Transfer

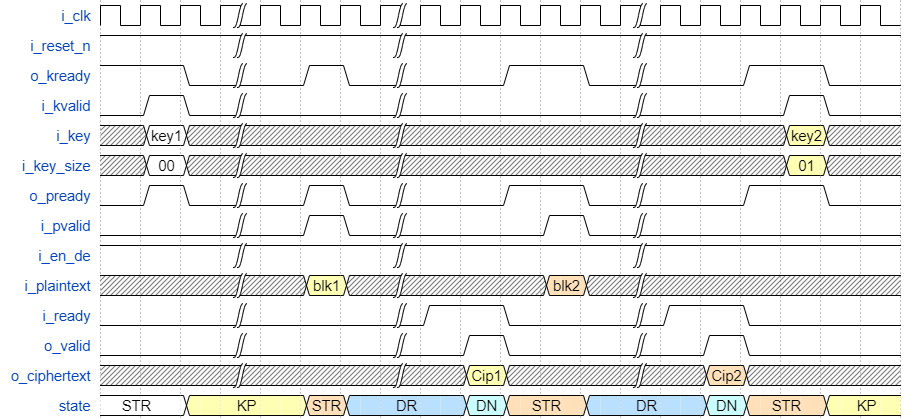


Figure 5: handshake signals waveform

# Hierarchy of Modules

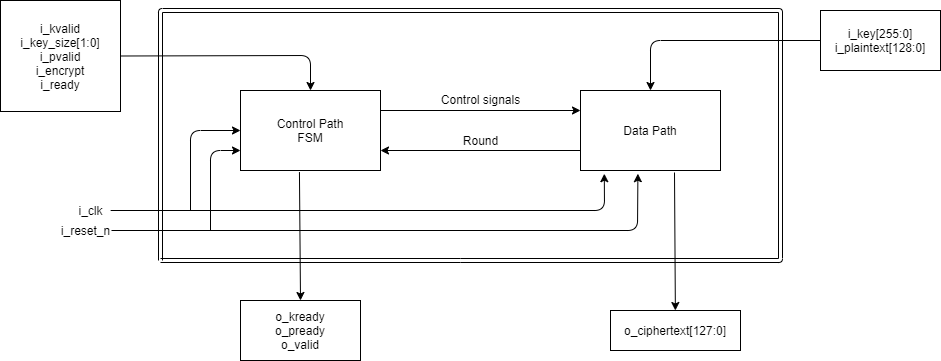


Figure 6: Hierarchy of Modules

## Control path

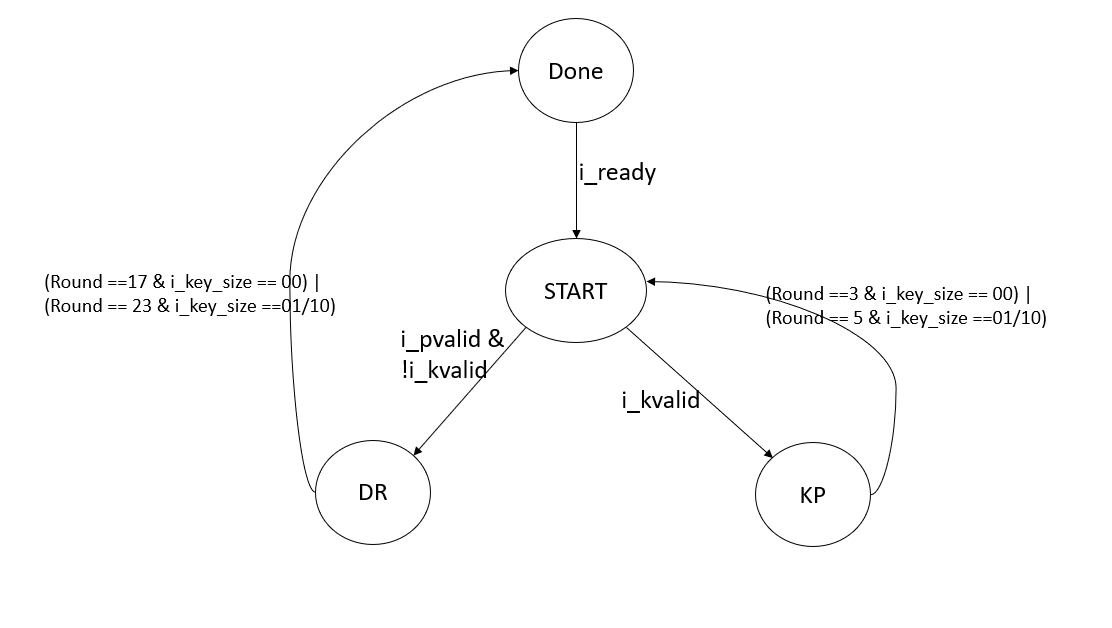


Figure 7: control path FSM state diagram

### Description:

|  |  |
| --- | --- |
| State  (state encoding) | Description |
| START  (00) | * It is the default state. * The outputs **i\_kready** and **i\_pready** are made HIGH * Priority is given to **i\_kvalid** to go to the **KEY\_PHASE** state, else if **i\_pvalid** then go to **DATA\_RANDOMIZATION** state. * **START** is the round – 1 of either key phase or data randomization phase depending on the valid ready handshake based on the priority. * The round -1 computations are done in this state |
| KEY\_PHASE(KP)  (01) | * This state is we generate the 128-bit keys KL, KR, KA and KB which are used in the DATA\_RANDOMIZATION state. * For a 128-bit key the KEY\_PHASE state is only for 3 rounds * For a 192/256-bit key the KEY\_PHASE state is for 5 rounds * It goes to the **START** state after the specified number of rounds. |
| DATA\_RANDOMIZATION  (DR)  (10) | * For 128-bit key this state is only for 17 rounds * For a 192/256-bit key this state is for 23 rounds * After the end of 17th round / 23th round the state machine enters **DONE** state. |
| DONE  (11) | * It lasts only for one cycle * In this state the cipher text is given as output and **o\_valid** is made high. * Next state is **START** if **i\_ready** is HIGH |

### Control Signals

* mux1\_sel
* mux2\_sel
* mux3\_sel
* mux4\_sel
* mux5\_sel[1:0]
* mux6\_sel[1:0]
* mux7\_sel[2:0]
* mux8\_sel[2:0]
* mux9\_sel[1:0]
* mux10\_sel[1:0]
* mux11\_sel
* mux12\_sel[1:0]
* mux13\_sel[1:0]
* mux14\_sel
* mux15\_sel

## Data path

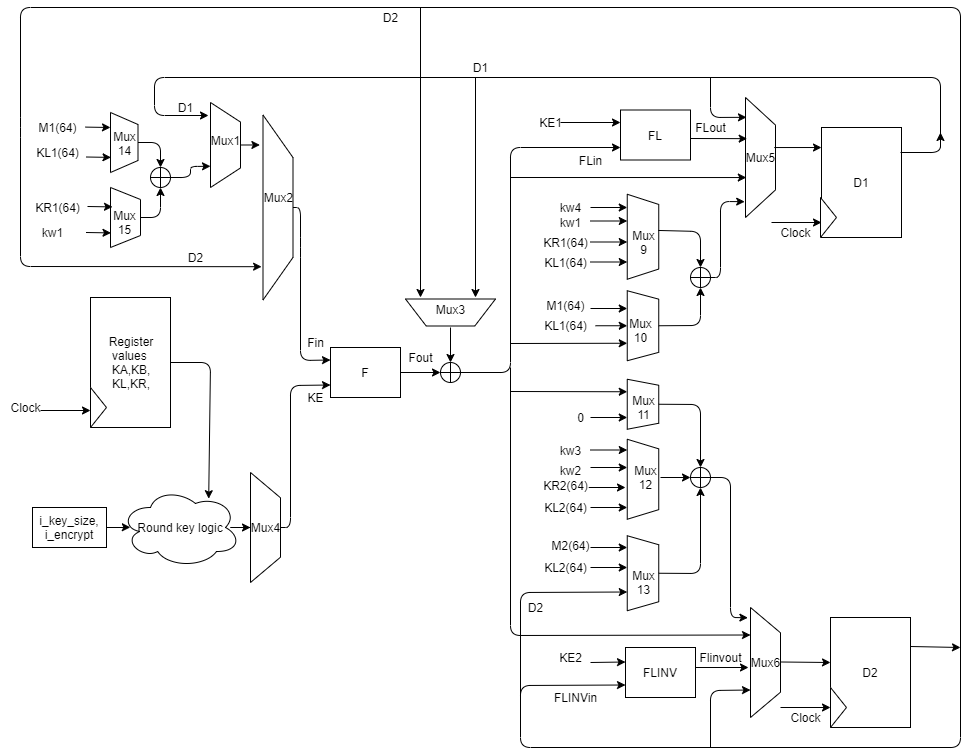


Figure 8: Main data-path

### Function – F

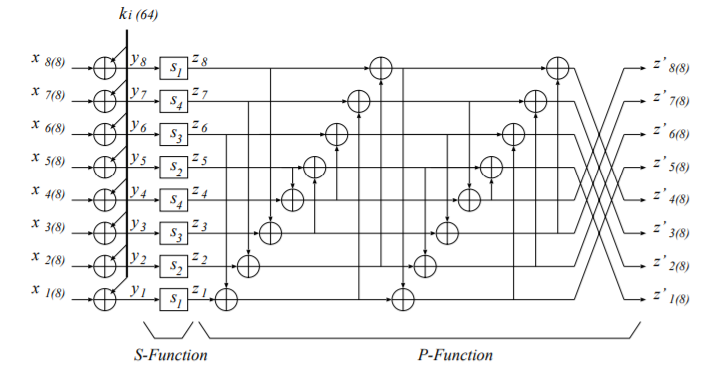


Figure 9: Function F

### Function – FL and Function – FLINV

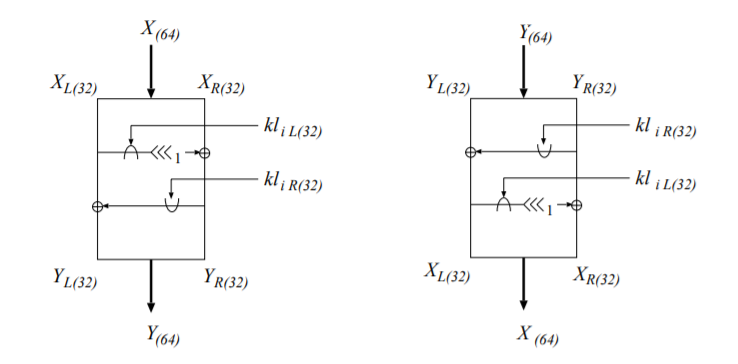


Figure 10: Functions FL and FLINV

### Key Generation for Function- FL and FLINV

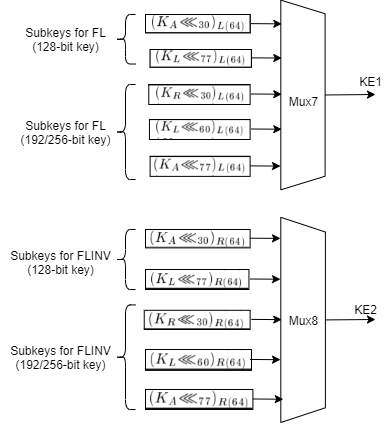


Figure 11: key generation for FL and FLINV

The design shown in figure-8 is the generalised structure that is used in the Key phase and data randomization Phase.

* Mux 1 selects between the Register value D1 and the combinational logic.
* Mux 2 selects between the output of Mux 1 and D2
* Mux 3 Selects between the register values D1 and D2
* Mux 4 Selects which KE has to be given to Function - F
* Mux 5 and 6 selects the value that the registers D1 and D2 have to be updated at the next clock edge

### Round Key logic

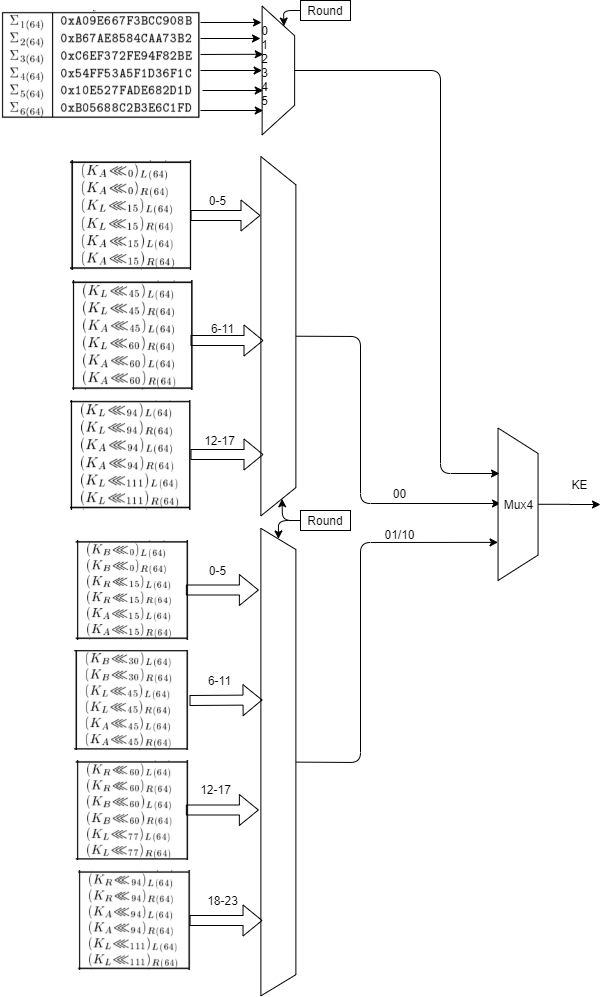


Figure 12: Round key logic

# Example Scenarios – KEY PHASE

## Scenario 1

Consider a case where I have just entered the START state and a valid ready handshake has taken place for the Key - K which is 128-bits, and the round(counter) has the value 0, The following figure shows the flow of data in the data-path for the first round.

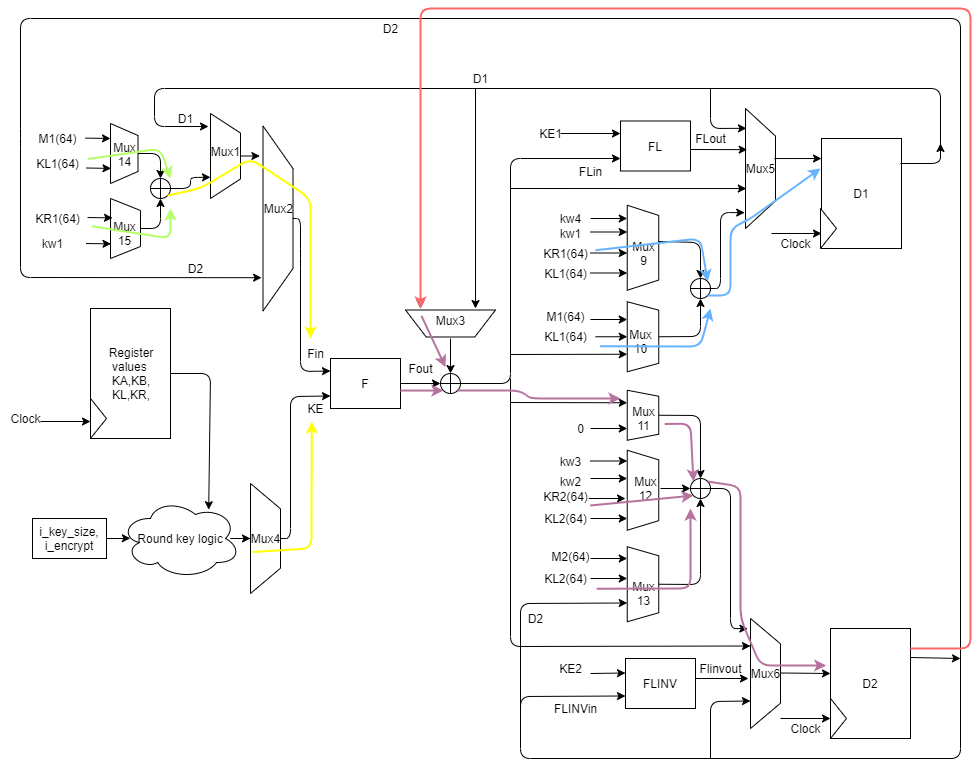


Figure 13: example 1

1. The input to the Function F is KL ^ KR, leftmost 64 bits, which is computed and this value is selected by Mux1, which flows through Mux2.
2. The sub key for this round is sigma1 which comes through Mux4.
3. These inputs are used for Function F and the output Fout is produced.
4. The current values in register D2 is 0, (cleared in the previous state)
5. D2 is selected by Mux3 and is XORed with Fout.
6. The value KL[127:64] ^ KR[127:64] calculated in point 1 is given as the next state input to the register D1 through Mux 5.
7. The values D2(0) ^ Fout ^ KL[63:0] ^ KR[63:0] is computed and given as next state input to the register D2 through Mux 6.
8. The values of D1 and D2 are updated in the coming clock edge and the round(counter) is incremented and we enter the KEY\_PHASE state for having a valid ready handshake in this state.

## Scenario 2

Now consider the second round and we are in the KEY\_PHASE state. The following figure shows the flow of data

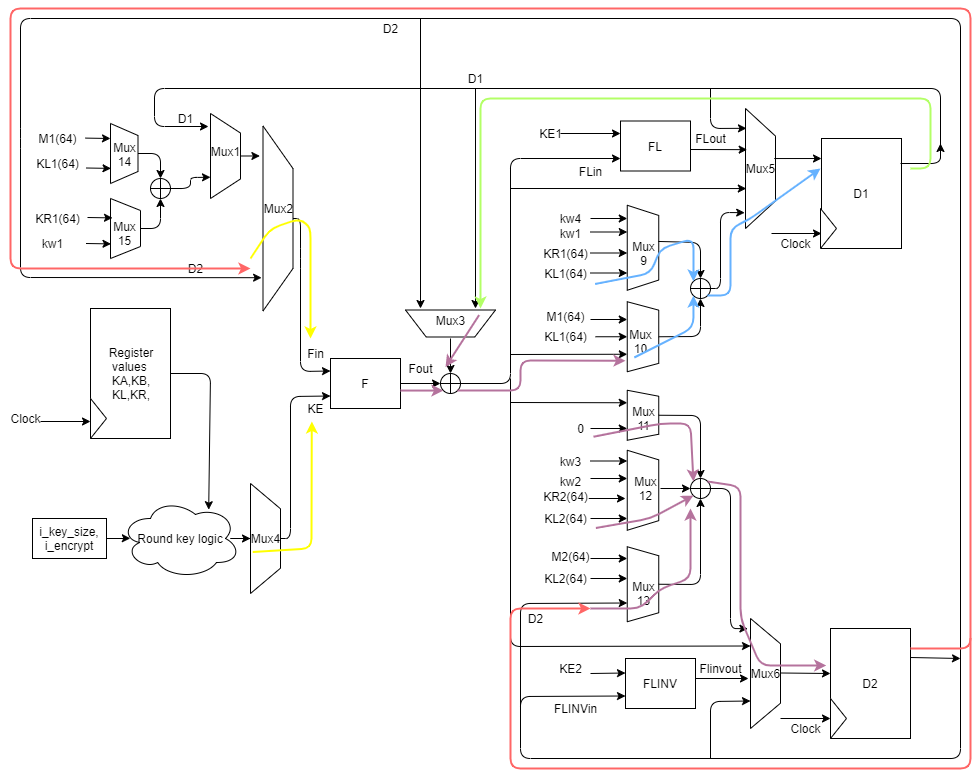


Figure 14: example 2

1. The input to Function F is the value stored in D2 which comes through Mux 2
2. The subkey KE, sigma2 to function F comes from Mux 4
3. The output Fout is generated
4. In this round the Fout is XORed with the register value stored in D1 (through Mux 3) and this is XORed with KL leftmost 64bits and is used as the next state value for D1 through Mux 5
5. The next state value for D2 is computed, which is D2 ^ KL[63:0]
6. These values are updated in the next clock edge and round(counter) is incremented.

## Scenario 3

Now consider the third round of KEY\_PHASE state.

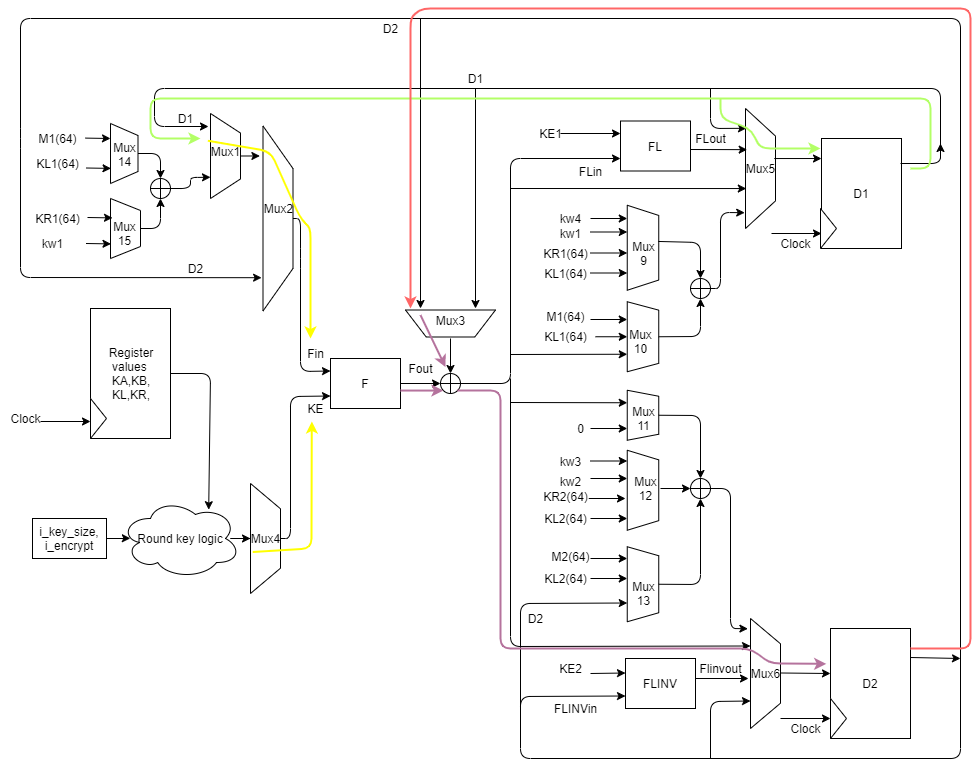


Figure 15: Example 3

1. The input to the Function F is current values of D1
2. The sub key for this round is sigma3 which comes through Mux4.
3. These inputs are used for Function F and the output Fout is produced.
4. D2 is selected by Mux3 and is XORed with Fout.
5. The value of register D1 is retained
6. The next state value of register D2 is the value computed in point – 4.
7. The values of D1 and D2 are updated in the coming clock edge and the round(counter) is incremented.

## Scenario 4

Now consider the fourth round. The following figure shows the flow of data

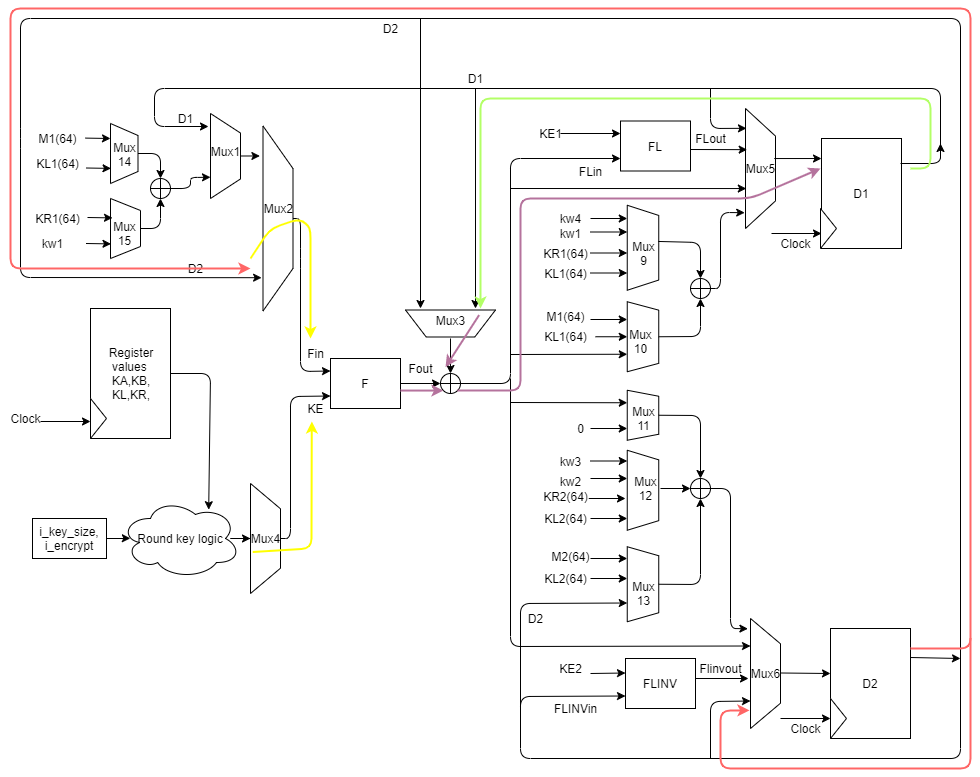


Figure 16: Example 4

1. The input to the Function F is current values of D2
2. The sub key for this round is sigma4 which comes through Mux4.
3. These inputs are used for Function F and the output Fout is produced.
4. D1 is selected by Mux3 and is XORed with Fout.
5. The value of register D2 is retained
6. The next state value of register D1 is the value computed in point – 4.
7. The values of D1 and D2 are updated in the coming clock edge and the round(counter) is incremented.

## Waveforms depicting the KEY PHASE state

* 128-bit key

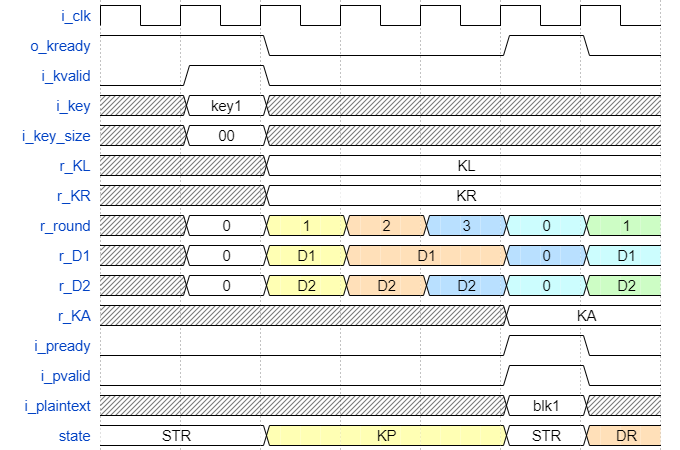


Figure 17: waveform for 128-bit key

* 192/256-bit key

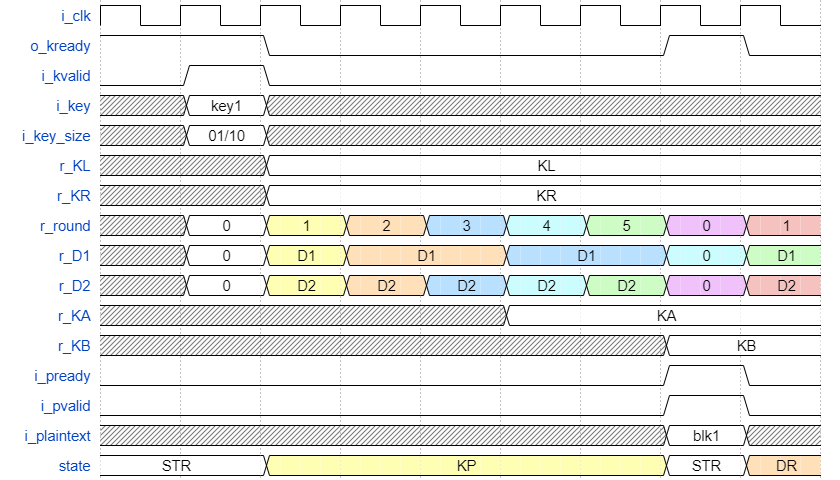


Figure 18: waveform for 192 and 256-bit key

# Example Scenarios – DATA RANDOMIZATION

## Scenario 1

Consider the state START and a valid ready handshake has taken place for the Plaintext, the following figure shows the data-path flow. This is the first round of data randomization.

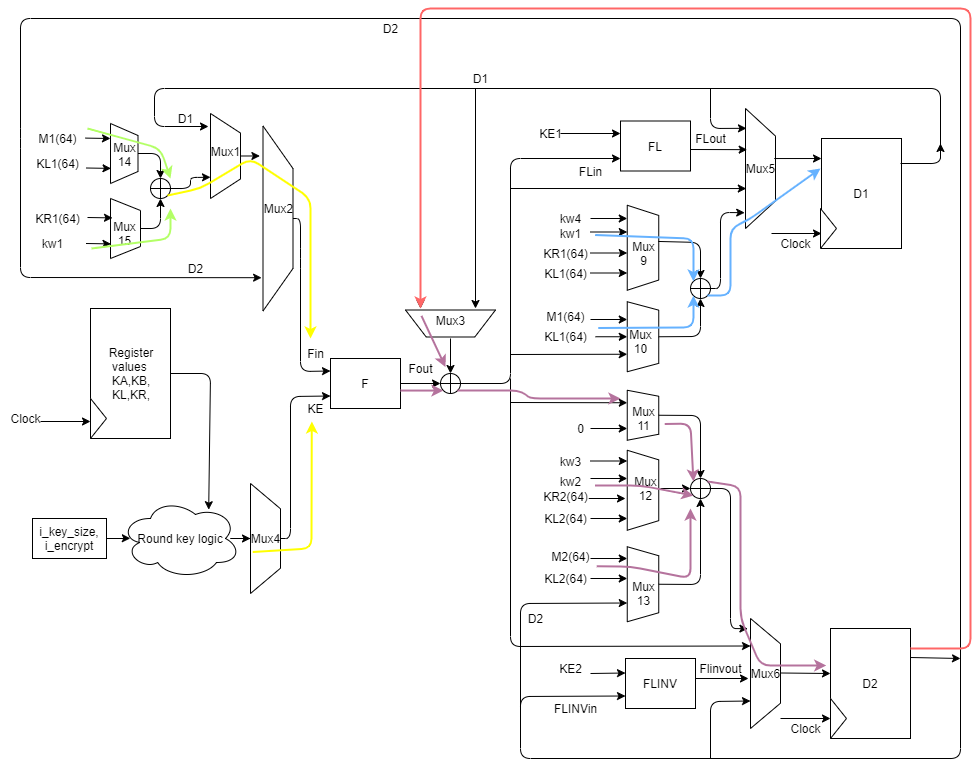


Figure 19: Example 1

1. The input to Function F is M[127:64]^kw1 (pre-whitening) which comes through Mux 2, its also the next state value of register D1
2. The subkey KE = k1 to function F comes from Mux 4
3. The output Fout is generated
4. Then Fout is XORed with D2(=0, cleared in the previous cycle)
5. The value computed in point-4 is XORed with M[63:0] and Kw2(pre-whitening) i.e., The state value of Register D2 is 0 ^ Fout ^ M2 ^ kw2.
6. At the next coming clock edge, the registers D1 and D2 are updated, and round is also incremented.

## Scenario 2

In the second round we enter the DATA\_RANDOMIZATION state, the following data flow happens

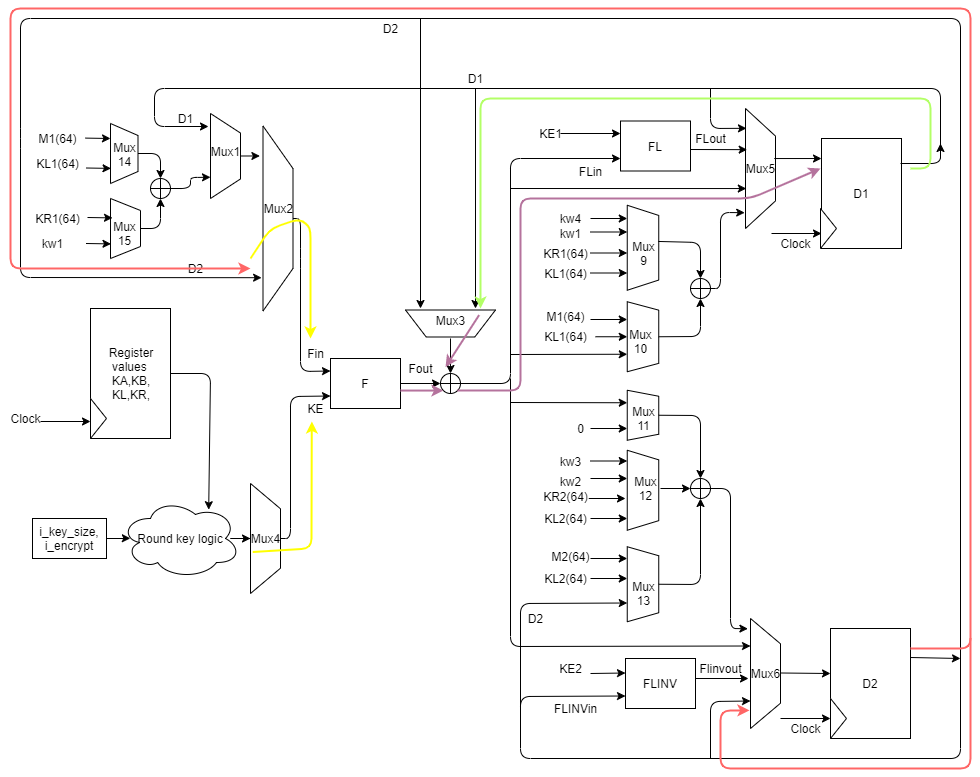


Figure 20: Example 2

1. In this round the Input to Function – F is Fin = D2
2. The subkey KE is k2 which comes through Mux4
3. The current state value of register D1 is XORed with Fout, which is the next state value of the register D1
4. The value of D2 is retained.
5. These values are sampled at the next clock edge and the round (counter) is also incremented.

## Scenario 3

Consider the round 6 of DATA RANDOMIZATION (round value (counter) = 5), where functions FL and FLINV are used

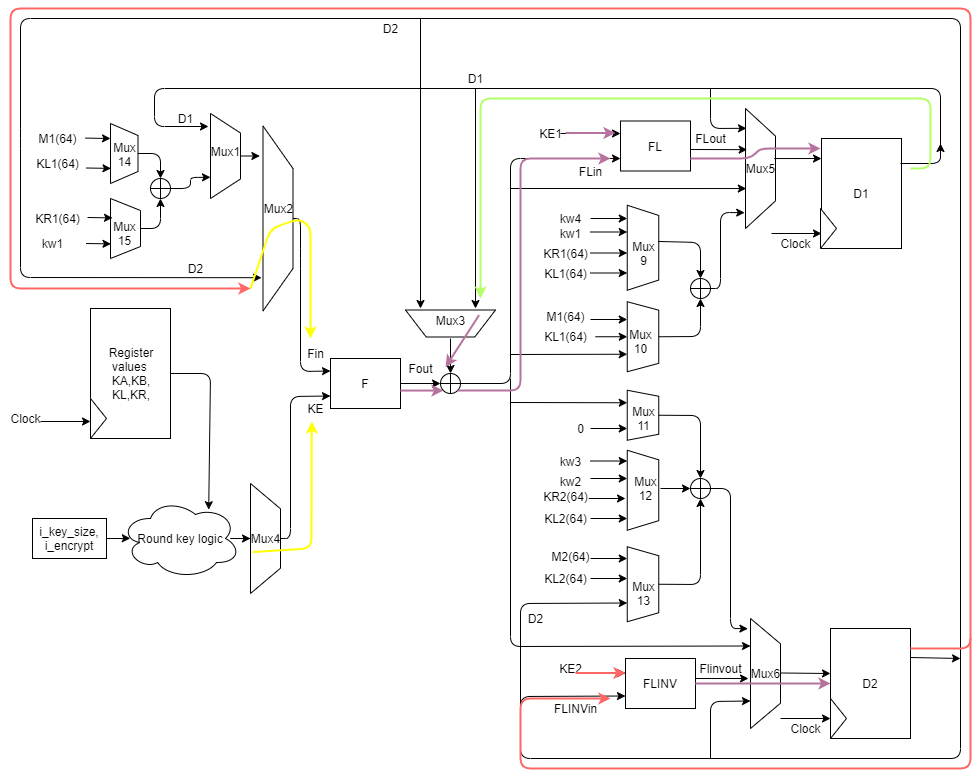


Figure 21: Example 3

1. In this round the Input to Function – F is Fin = D2
2. The subkey KE is k6 which comes through Mux4
3. These inputs are used to compute Fout
4. The Fout is XORed with Register value D1
5. The value computed in point 4 is the FLin, the subkey KE1 flows through Mux 7
6. FLINVin is the register value D2, the subkey KE2 comes through Mux 8
7. The Flout and FLINV out are the next state values of registers D1 and D2 respectively
8. At the next coming clock edge, the registers D1 and D2 are updated

## Scenario 4

Consider the last round of DATA RANDOMIZATION where post-whitening is involved.

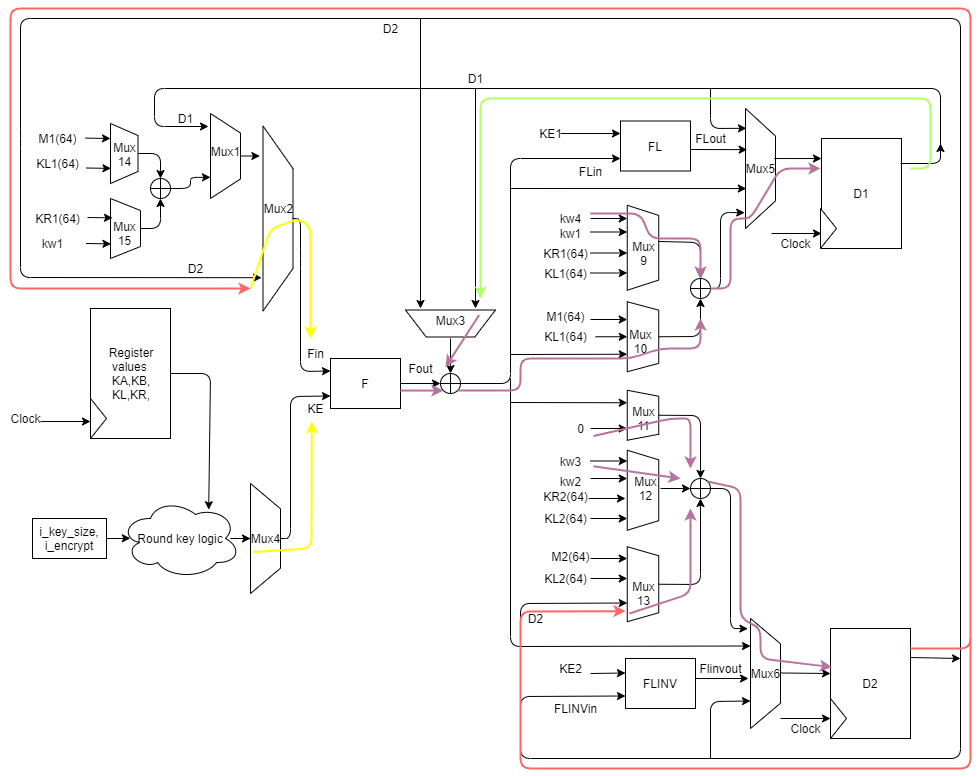


Figure 22: Example 4

1. In this round the Input to Function – F is Fin = D2
2. The subkey KE is k6 which comes through Mux4
3. These inputs are used to compute Fout
4. The Fout is XORed with Register value D1
5. The value Fout ^ D1 ^ kw4 is the next state value for the register D1
6. The value D2 ^ kw3 is the next state value for the register D2
7. Cipher text = {D2 ^ kw3, Fout ^ D1 ^ kw4} is given as the input to register C
8. In the coming clock edge, the values computed get sampled, and we enter the DONE state.

## Waveform depicting the DATA\_RANDOMIZATION state

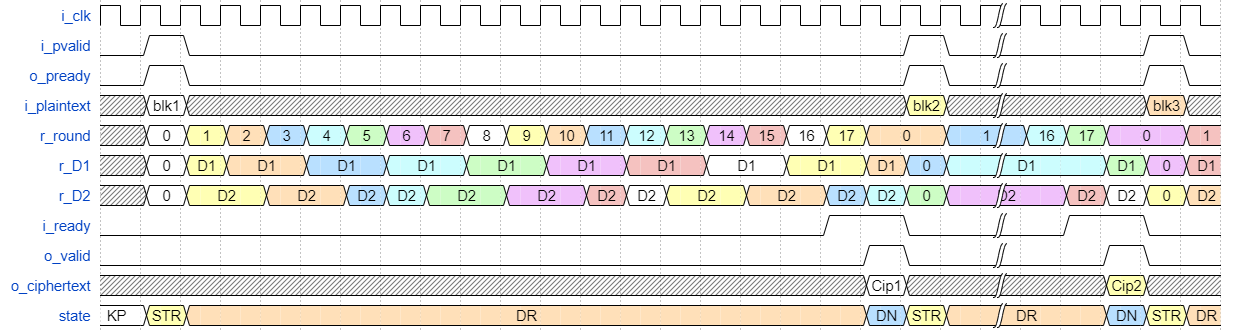


Figure 23: waveform 128-bit key

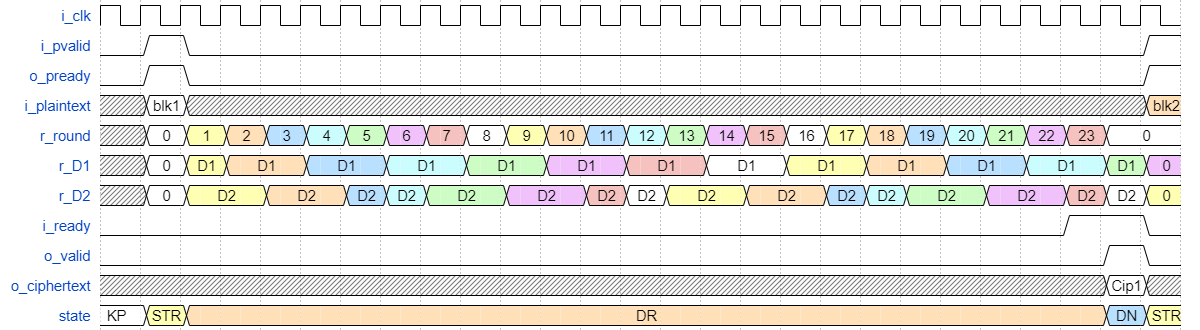


Figure 24: waveform for 192/256 - bit key

### Number of cycles for key phase

1. For a 128-bit key the key phase is for 4 cycles.
2. For a 192/256-bit key the key phase is for 6 cycles.

### Number of cycles for data randomization

1. For a 128-bit key the DATA\_RANDOMIZATION state is for 18 cycles, one extra cycle for the user to take the valid ciphertext.
2. For a 192/256-bit key the DATA\_RANDOMIZATION state is for 24 cycles, one extra cycle for the user to take the valid ciphertext.