EE 4313
Computer Engg. Design Project 1
Lecture 2

Hardware Security

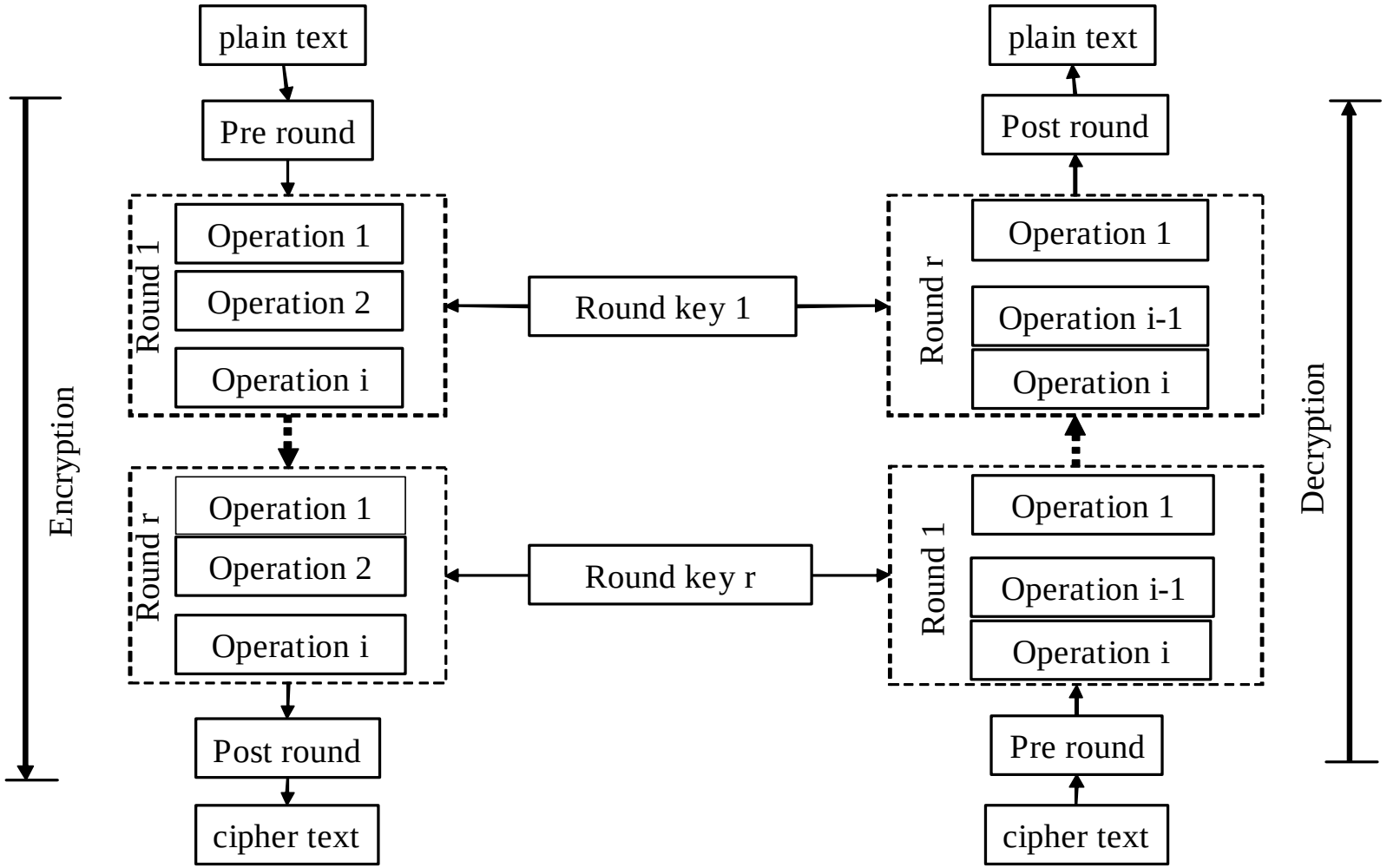
- Security and Privacy imply support for
  - Data confidentiality
  - Data integrity
  - Authentication
  - Non-repudiation
  - Symmetric Key Crypto is the building block

Data confidentiality

- Symmetric block cipher
  - Encrypt (plaintext block, key)= ciphertext block
  - Decrypt (ciphertext block, key)= plaintext block
  - Encryption key = Decryption key
  - Data Encryption Standard (DES)
    - 64-bit plaintext block, 56-bit secret key
  - Advanced Encryption Standard (AES)
    - 128-bit plaintext block, 128-bit secret key
  - RC5
    - 64-bit plaintext block, 128-bit key

RC5 Symmetric block cipher

Symmetric Block Cipher



RC5

- Designed by Ronald Rivest, MIT
- Design principles
  - Simple
  - Secure
  - Fast in hardware
  - Fast in software
- RC5
  - 64-bit input; 64 bit output
  - 12 (encryption/decryption) rounds
    - Each round uses two round keys
  - 16 bytes in user secret key
  - Data dependent rotate

RC5 Components

- Encryption
- Decryption
- Decryption = Encryption^{-1}
- Round key generation

RC5 Encryption

```
A = A + S[0];
B = B + S[1];
for i = 1 to 12 do
    A = ((A xor B) <<< B) + S[2×i];
    B = ((B xor A) <<< A) + S[2×i + 1];
```

- A and B are 32-bit halves of input plaintext
- S[0], S[1], S[2]…S[25] are the 26 round keys
  - 32-bits each → How many key bits?
  - round key generation: 128-bit user key → 26 32-bit round keys

RC5 Decryption

```
for i = 12 downto 1 do
   B = ((B - S[2×i +1]) >>> A) xor A;
   A = ((A - S[2×i]) >>> B) xor B;
B = B - S[1];
A = A - S[0];
```

- Inverse of encryption
- A and B are 32-bit halves of input ciphertext
- S[25], S[24], S[23]... are round keys used in that order

RC5 Decryption = RC5 Encryption^{-1}

Encryption	Decryption
A0 = A + S[0]	A = A0 - S[0]
B0 = B + S[1]	B = B0 - S[1]
A1 = ((A0 xor B0) <<< B0) + S[2]	A0 = ((A1 – S[2])>>> B0) xor B0
B1 = ((B0 xor A1) <<< A1) + S[3]	B0 = ((B1 - S[3]) >>> A1) xor A1
A2 = ((A1 xor B1) <<< B1) + S[4]	A1 = ((A2 - S[4]) >>> B1) xor B1
B2 = ((B1 xor A2) <<< A2) + S[5]	B1 = ((B2 – S[5]) >>> A2) xor A2
⋮	⋮
A12=((A11 xor B11)<<<B11)+S[24]	A11=((A12- S[24])>>>B11)xorB11
B12=((B11 xor A12)<<<A12)+S[25]	B11=((B12–S[25])>>>A12)xor A12

Lab 2: RC5 Encryption/Decryption Iterative Architecture

- Implement RC5 encryption without pre-round
  - Input: 64-bit plaintext (A, B 32 bit inputs each)
  - Output: 64-bit ciphertext (A, B are 32 bit outputs each)
  - S [2], S[3], …, S[24], S[25] are 32-bit round keys (these will be provided)
- Implement RC5 decryption without post-round
  - Input: 64-bit ciphertext (A, B are 32 bit input words)
  - Output: 64-bit plaintext (A, B are 32 bit output words)
  - S [25], S[24], …, S[3], S[2] are 32-bit round keys (these will be provided)

Lab 2: RC5 Encryption (w/o pre-round)

- 64-bit data input, 64-bit output
- Round keys S[2],..., S[25] are fixed and hard coded in hardware

```
A = A + S[0];
B = B + S[1];
for i=1 to 12 do {
  A = ((A XOR B) <<< B) + S[2×i];
  B = ((B XOR A) <<< A) + S[2×i+1];
}
```

RC5 Encryption: Entity Definition

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL; --use CONV_INTEGER
ENTITY rc5_enc IS
 PORT  (
  clr: IN STD_LOGIC;  -- asynchronous reset
  clk: IN STD_LOGIC;  -- Clock signal
  din: IN STD_LOGIC_VECTOR(63 DOWNTO 0);--64-bit input
  dout: OUT STD_LOGIC_VECTOR(63 DOWNTO 0) --64-bit output
  );
END rc5_enc;
```

RC5 Encryption: Architecture Description (Internal Signals)

```
ARCHITECTURE rtl OF rc5_enc IS
  --round counter
  SIGNAL i_cnt: STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL ab_xor: STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL a_rot: STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL a: STD_LOGIC_VECTOR(31 DOWNTO 0);
  --register to store value A
  SIGNAL a_reg: STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL ba_xor: STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL b_rot: STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL b: STD_LOGIC_VECTOR(31 DOWNTO 0);
  --register to store value B
  SIGNAL b_reg: STD_LOGIC_VECTOR(31 DOWNTO 0);
```

RC5 Encryption: Hardcode round keys

```
TYPE rom IS ARRAY (2 TO 25) OF
  STD_LOGIC_VECTOR(31 DOWNTO 0); CONSTANT skey
  : rom:=rom'(X"46F8E8C5",X"460C6085","70F83B8A",
  X"284B8303", X"513E1454", "F621ED22",
  X"3125065D",X"11A83A5D",X"D427686B",
  X"713AD82D", X"4B792F99", X"2799A4DD",
  X"A7901C49", X"DEDE871A", X"36C03196",
  X"A7EFC249", X"61A78BB8", X"3B0A1D2B",
  X"4DBFCA76", X"AE162167", X"30D76B0A",
  X"43192304", X"F6CC1431", X"65046380");
```

RC5 Encryption: round operation step a

A=((A XOR B)<<<B) + S[2×i]

```
BEGIN
  -- A=((A XOR B)<<<B) + S[2×i];
 ab_xor <= a_reg XOR b_reg;
 WITH b_reg(4 DOWNTO 0) SELECT
   a_rot<=ab_xor(30 DOWNTO 0) & ab_xor(31) WHEN
"00001",
                ......
              ab_xor WHEN OTHERS;
 a<=a_rot + skey(CONV_INTEGER(i_cnt & '0')); --S[2×i]
```

You wrote this model in Lab 1

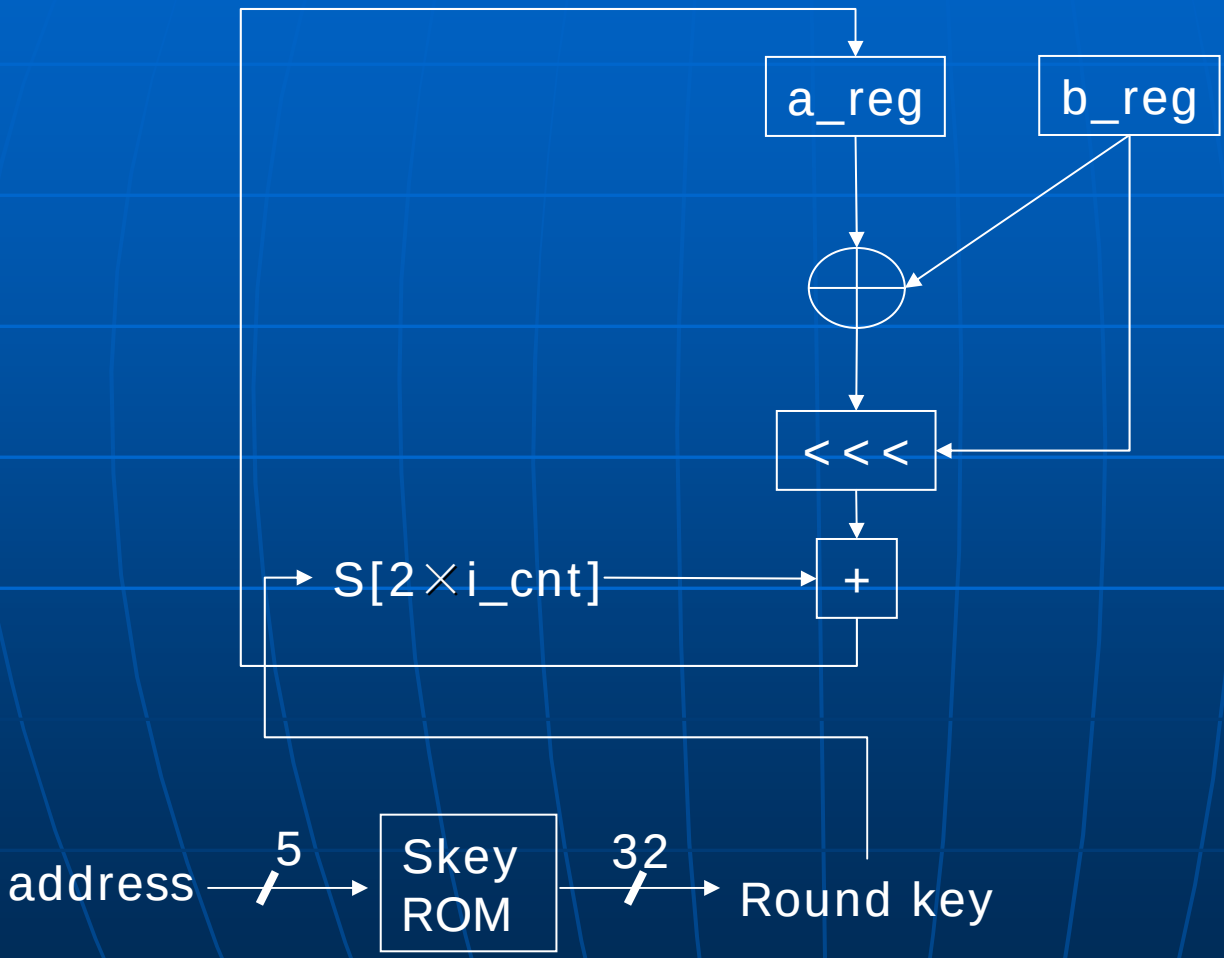
RC5 Encryption: round operation step b

B=((B XOR A)<<<A) + S[2×i+1]

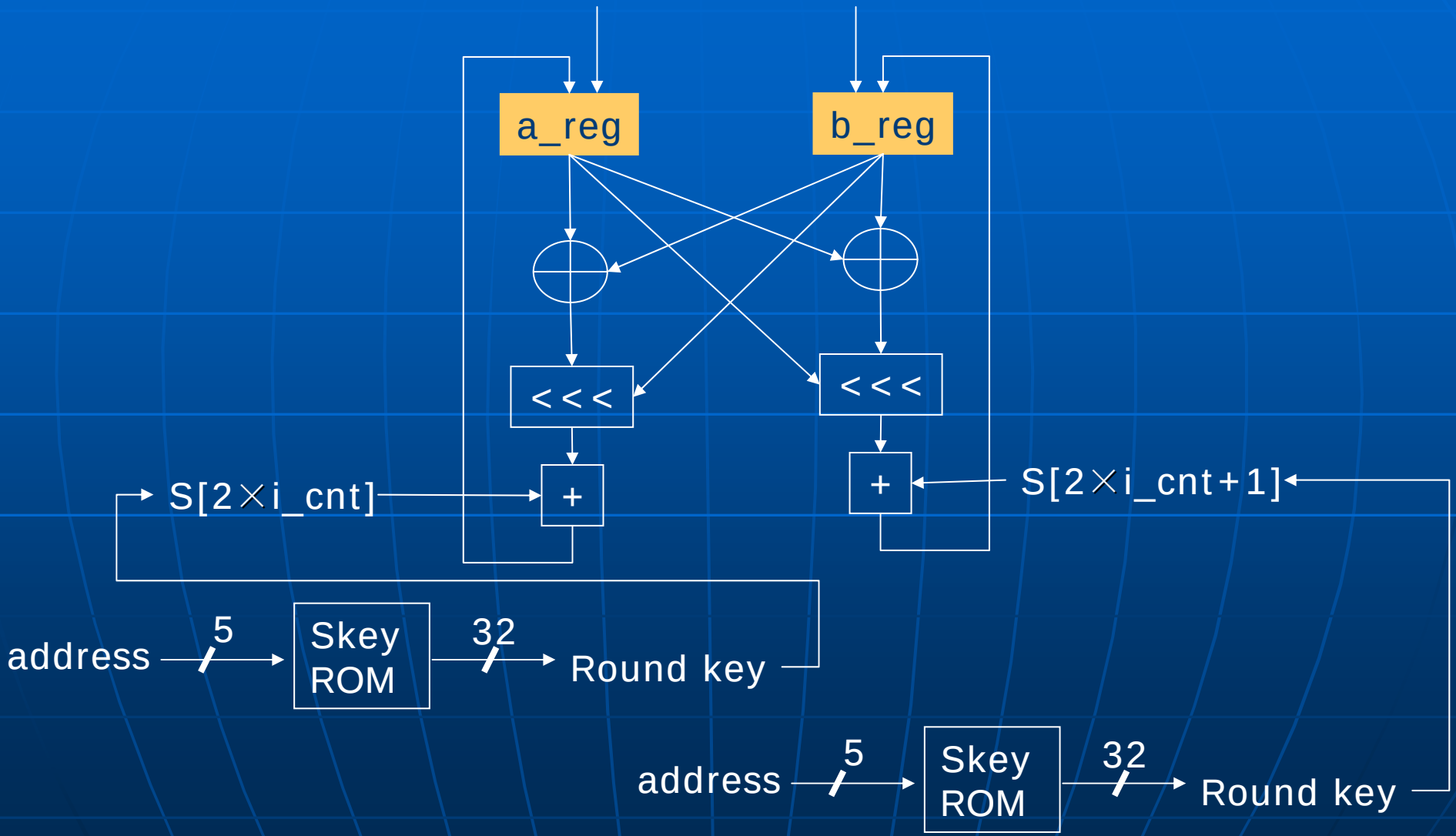
```
ba_xor <= b_reg XOR a;
  WITH a(4 DOWNTO 0) SELECT
    b_rot<=ba_xor(30 DOWNTO 0) & ba_xor(31) WHEN
"00001",
                  ......
               ba_xor WHEN OTHERS;
 b<=b_rot+skey(CONV_INTEGER(i_cnt & '1'));--S[2×i+1]
```

Modify the model you wrote in Lab 1

RC5 data path: step a

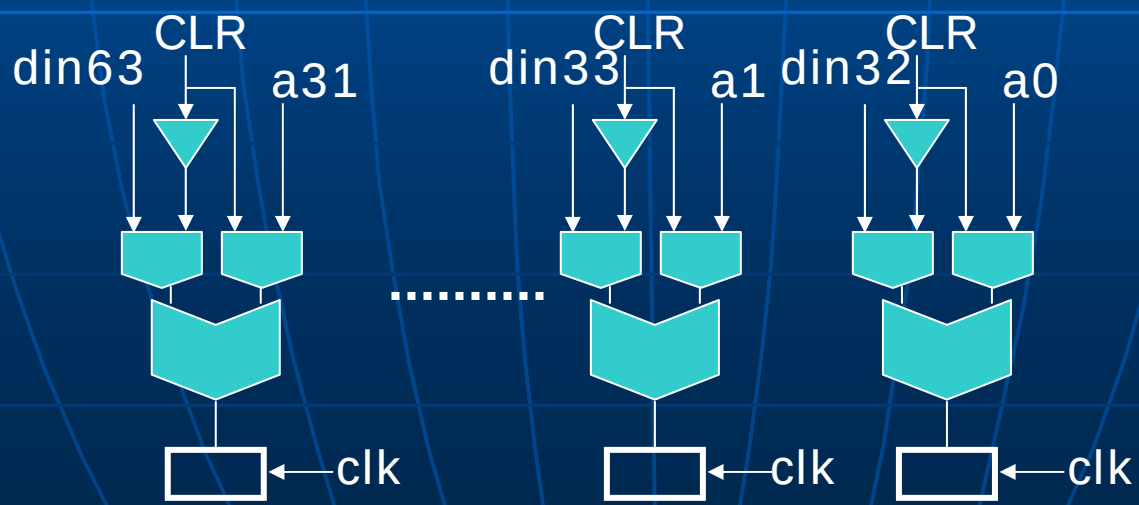


RC5 Encryption Data path



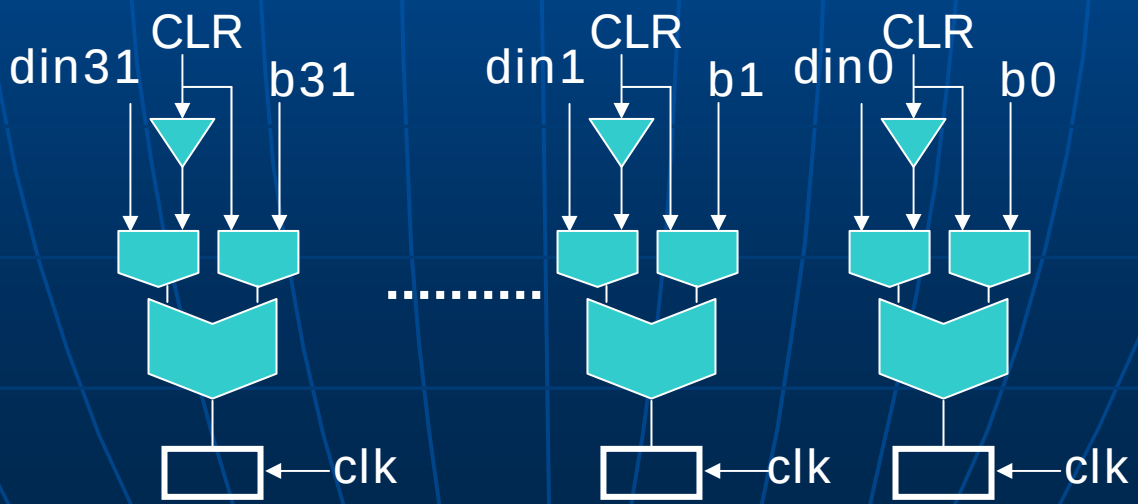
RC5 Encryption: Sequential Circuit Model Example 1 (32-bit register)

```
-- Register A
PROCESS(clr, clk)  BEGIN
  IF(clr='0') THEN
      a_reg<=din(63 DOWNTO 32);
  ELSIF(clk'EVENT AND clk='1') THEN
      a_reg<=a;
  END IF;
END PROCESS;
```



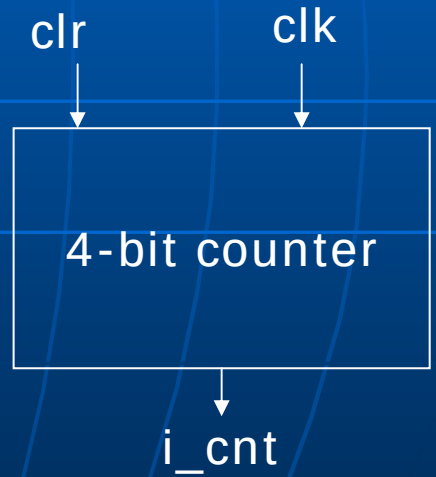
RC5 Encryption: Sequential Circuit Example 2 (32-bit register)

```
-- Register B
   PROCESS(clr, clk)  BEGIN
      IF(clr='0') THEN
         b_reg<=din(31 DOWNTO 0);
      ELSIF(clk'EVENT AND clk='1') THEN
         b_reg<=b;
      END IF;
   END PROCESS;
```



RC5 Encryption: Sequential Circuit Modeling Example 3 (4-bit counter)

```
PROCESS(clr, clk)  BEGIN
  IF(clr='0') THEN
     i_cnt<="0001";
   ELSIF(clk'EVENT AND clk='1') THEN
      IF(i_cnt="1100") THEN
        i_cnt<="0001";
      ELSE
        i_cnt<=i_cnt+'1';
      END IF;
    END IF;
END PROCESS;
```



RC5 encryption: assign results to output

```
dout<=a_reg & b_reg;
END rtl;
```

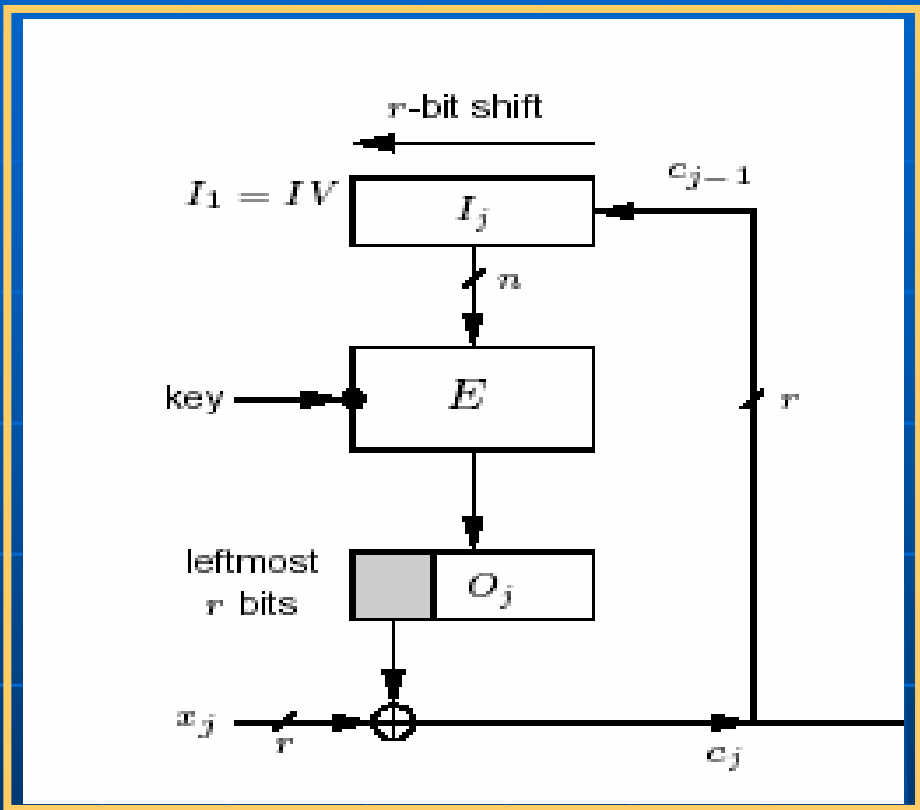
Lab 2: RC5 Encryption/Decryption Iterative Architecture

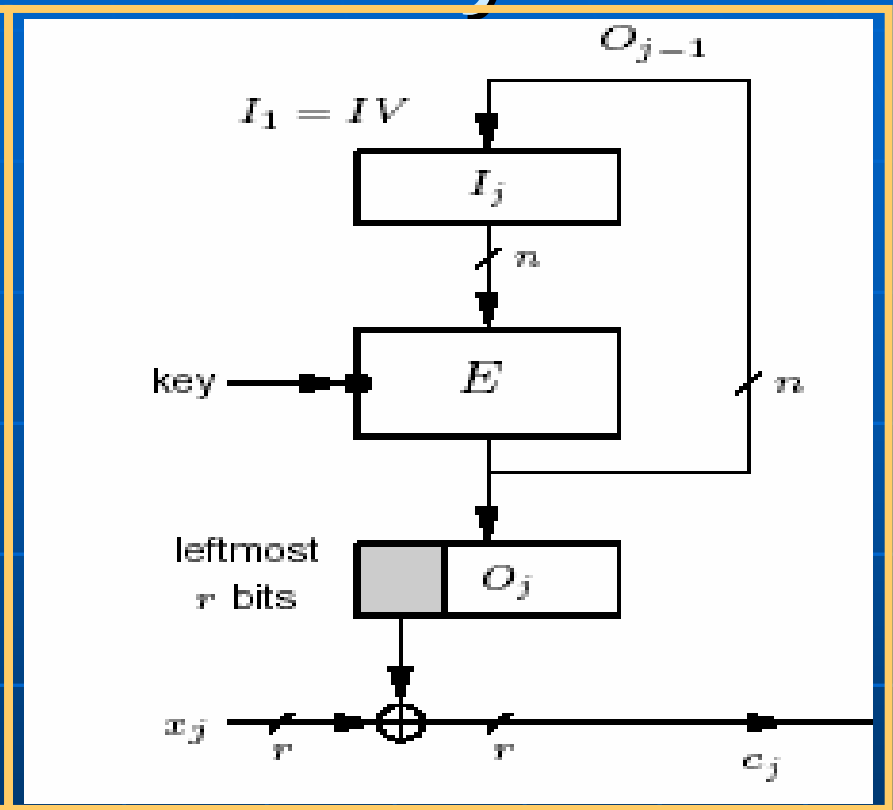
- Implement RC5 encryption without pre-round
  - Input: 64-bit plaintext (A, B are 32 bit input words)
  - Output: 64-bit ciphertext (A, B are 32 bit output words
  - S [2], S[3], …, S[24], S[25] are 32-bit round keys (these will be provided)
- Implement RC5 decryption without post-round
  - Input: 64-bit ciphertext (A, B are 32 bit input words)
  - Output: 64-bit plaintext (A, B are 32 bit output words)
  - S [25], S[24], …, S[3], S[2] are 32-bit round keys (these will be provided)
- Pre-lab: show your design for decryption.

Pre-Lab for Lab 2: RC5 Encryption/Decryption Iterative Architecture (20% of your Lab 2 grade)

- Implement RC5 encryption without pre-round
- Implement RC5 decryption without post-round
- Pre-lab
  - show your design for encryption and decryption (I showed you the design for encryption)
  - Write the entity description for encryption (I gave it to you) and decryption
  - Explain in detail how your encryption and decryption data paths will implement the RC5 encryption and decryption algorithms.

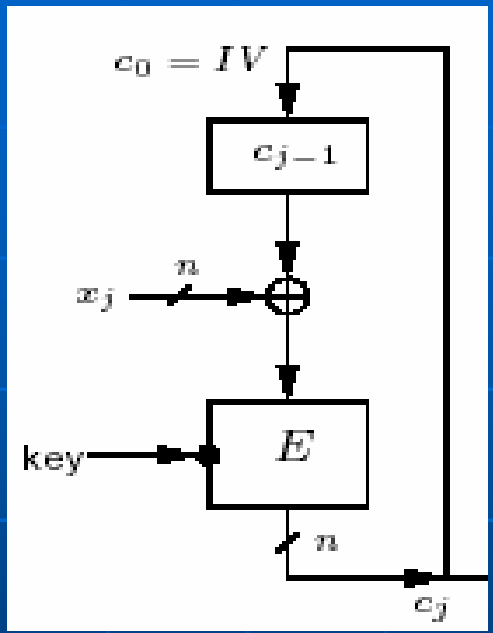
Data Confidentiality





- Stream ciphers
  - Key stream generator; encryption is a simple xor operation
  - Cipher feedback mode of DES, AES, RC5 etc..
  - Output feedback mode of DES, AES, RC5 etc..

Data Integrity



- Message authentication code
  - Detect modification to messages
  - Cipher block chaining mode of AES,DES,RC5…

VHDL Case Statement

- Sequential statement
- Can only be used inside a process

O =(A XOR B) <<<B + SKEY

```
LIBRARY    IEEE;
USE        IEEE.STD_LOGIC_1164.ALL;
ENTITY     leftrotate IS
  PORT (a: in STD_LOGIC_VECTOR(31 DOWNTO 0);
        b: in STD_LOGIC_VECTOR(31 DOWNTO 0);
        skey: out STD_LOGIC_VECTOR(31 DOWNTO 0);
        o: out STD_LOGIC_VECTOR(31 DOWNTO 0));
END leftrotate;

ARCHITECTURE rtl OF leftrotate IS
SIGNAL ab_xor: STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL ab_rot: STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
ab_xor <= a XOR b;
```

O =(A XOR B) <<<B + SKEY

```
PROCESS(b, ab_xor)
BEGIN
CASE b(4 DOWNTO 0) IS
 WHEN “00001”=> ab_rot<= ab_xor(30 DOWNTO 0)&ab_xor(31);
 WHEN “00010”=> ab_rot<=ab_xor(29 DOWNTO 0)&ab_xor(31DOWNTO
30);
 WHEN "00011”=> ab_rot<= ab_xor(28 DOWNTO 0) & ab_xor(31 DOWNTO
29);
 WHEN "00100“=> ab_rot<= ab_xor(27 DOWNTO 0) & ab_xor(31 DOWNTO
28);
 WHEN "00101“=> ab_rot<= ab_xor(26 DOWNTO 0) & ab_xor(31 DOWNTO
27);
 WHEN "00110“=> ab_rot<= ab_xor(25 DOWNTO 0) & ab_xor(31 DOWNTO
26);
 WHEN "00111" => ab_rot<= ab_xor(24 DOWNTO 0) & ab_xor(31 DOWNTO
25);
 WHEN "01000" => ab_rot<= ab_xor(23 DOWNTO 0) & ab_xor(31 DOWNTO
24);
 WHEN "01001" => ab_rot<= ab_xor(22 DOWNTO 0) & ab_xor(31 DOWNTO
23);
```

O =(A XOR B) <<<B + SKEY

```
WHEN "01010" => ab_rot<= ab_xor(21 DOWNTO 0) & ab_xor(31 DOWNTO
22);
 WHEN "01011" => ab_rot<= ab_xor(20 DOWNTO 0) & ab_xor(31 DOWNTO
21);
 WHEN "01100" => ab_rot<= ab_xor(19 DOWNTO 0) & ab_xor(31 DOWNTO
20);
 WHEN "01101" => ab_rot<= ab_xor(18 DOWNTO 0) & ab_xor(31 DOWNTO
19);
 WHEN "01110" => ab_rot<= ab_xor(17 DOWNTO 0) & ab_xor(31 DOWNTO
18);
 WHEN "01111" => ab_rot<= ab_xor(16 DOWNTO 0) & ab_xor(31 DOWNTO
17);
 WHEN "10000" => ab_rot<= ab_xor(15 DOWNTO 0) & ab_xor(31 DOWNTO
16);
WHEN "10001" =>ab_rot<= ab_xor(14 DOWNTO 0) & ab_xor(31 DOWNTO
15);
WHEN "10010" =>ab_rot<= ab_xor(13 DOWNTO 0) & ab_xor(31 DOWNTO
14);
WHEN "10011" =>ab_rot<= ab_xor(12 DOWNTO 0) & ab_xor(31 DOWNTO
13);
```

O =(A XOR B) <<<B + SKEY

```
WHEN "10100" =>ab_rot<= ab_xor(11 DOWNTO 0) & ab_xor(31 DOWNTO
12);
WHEN "10101" =>ab_rot<= ab_xor(10 DOWNTO 0) & ab_xor(31 DOWNTO
11);
WHEN "10110" =>ab_rot<= ab_xor(9 DOWNTO 0) & ab_xor(31 DOWNTO
10);
WHEN "10111" =>ab_rot<= ab_xor(8 DOWNTO 0) & ab_xor(31 DOWNTO
9);
WHEN "11000" =>ab_rot<= ab_xor(7 DOWNTO 0) & ab_xor(31 DOWNTO
8);
WHEN "11001" =>ab_rot<= ab_xor(6 DOWNTO 0) & ab_xor(31 DOWNTO
7);
WHEN "11010" =>ab_rot<= ab_xor(5 DOWNTO 0) & ab_xor(31 DOWNTO
6);
WHEN "11011" =>ab_rot<= ab_xor(4 DOWNTO 0) & ab_xor(31 DOWNTO
5);
WHEN "11100" =>ab_rot<= ab_xor(3 DOWNTO 0) & ab_xor(31 DOWNTO
4);
```

O =(A XOR B) <<<B + SKEY

```
WHEN "11101" =>ab_rot<= ab_xor(2 DOWNTO 0) & ab_xor(31 DOWNTO
3);
WHEN "11110" =>ab_rot<= ab_xor(1 DOWNTO 0) & ab_xor(31 DOWNTO
2);
WHEN "11111" =>ab_rot<= ab_xor(0) & ab_xor(31 DOWNTO 1);
WHEN OTHERS =>ab_rot<=ab_xor;
END CASE;
END PROCESS;
O <=ab_rot+skey
END rtl;
```