Elliptic Curve Cryptography on FPGA

Presentation By

Name - Sunny Kumar Pandit
ROll - CSE/22105/959
Team - ECC2

Introduction

- **Cryptography** is the practice of securing data through encoding, ensuring confidentiality, integrity, and authenticity.
- **Elliptic Curve Cryptography (ECC)** offers high security with smaller key sizes, making it ideal for modern lightweight cryptographic systems.
- **FPGAs** provide reconfigurable hardware with parallel processing capabilities, enabling efficient and fast implementation of cryptographic algorithms.
- This project aims to **design and implement ECC on FPGA** to achieve secure and resource-efficient cryptographic processing.

Elliptic Curve Cryptography

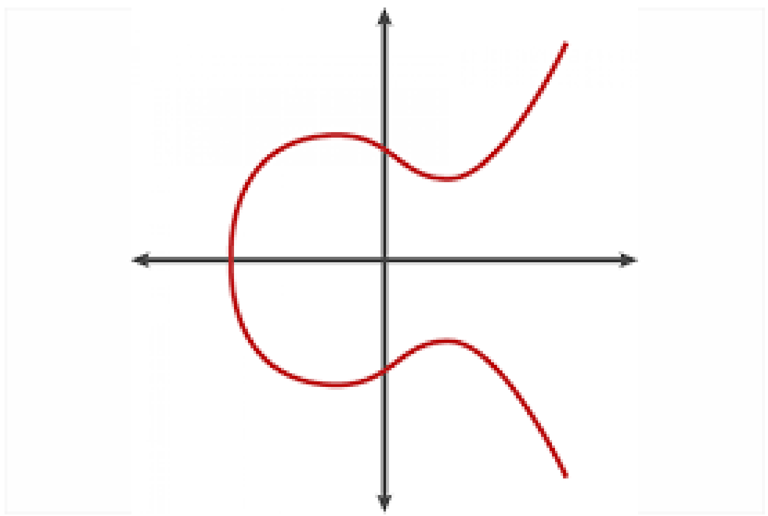
What is ECC ?

- ECC (Elliptic Curve Cryptography) is an asymmetric cryptographic method based on the algebraic structure of elliptic curves over finite fields.

- Uses the equation:

$$y^2 = x^3 + ax + b$$

(with the condition $4a^3 + 27b^2 \neq 0$ to ensure a valid curve)



- Elliptic Curve Cryptography (ECC) is not itself an encryption/decryption algorithm, but rather a framework used to generate keys securely using elliptic curve mathematics. These keys are then used in specific cryptographic protocols like ElGamal.

Why is ECC difficult to break ?

Let E be an elliptic curve defined over a finite field Fp. Let P be a point in E(Fp), and suppose that P has prime order n. Then the cyclic subgroup of E(Fp) generated by P :

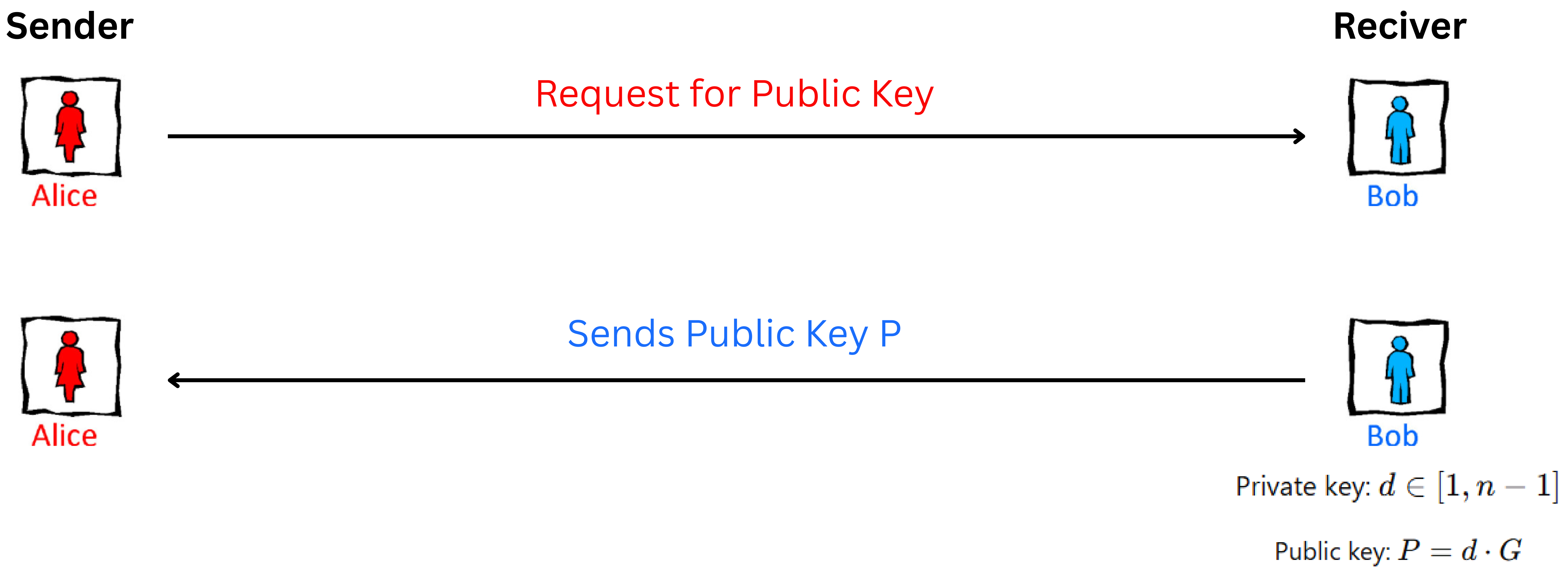
$$\langle P \rangle = \{\infty, P, 2P, 3P, \ldots, (n-1)P\}.$$

The prime p, the equation of the elliptic curve E, and the point P and its order n, are the public domain parameters. A private key is an integer d that is selected uniformly at random from the interval [1,n − 1], and the corresponding public key is Q = d P.

The problem of determining d given the domain parameters and Q is the elliptic curve discrete logarithm problem (ECDLP).

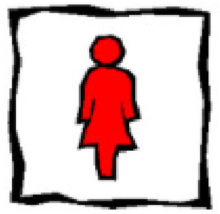
Working of ElGamal with ECC

- Elliptic curve E over a finite field $\mathbb{F}_p$
- Base point $G \in E$ of large prime order n



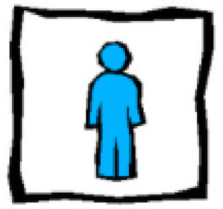
Sender

Reciver



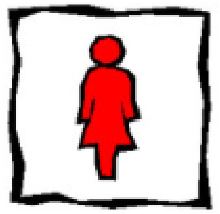
Alice

Sends Ciphertext (C1, C2)



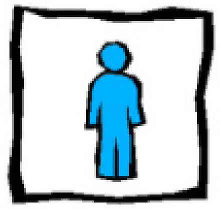
Bob

- Alice picks a random ephemeral key $k \in [1, n-1]$
- She computes:
  - $C_1 = k \cdot G$
  - $C_2 = M + k \cdot P$



Alice

Communication Done



Bob

He computes:

- $d \cdot C_1 = d \cdot (k \cdot G) = k \cdot (d \cdot G) = k \cdot P$
- $M = C_2 - k \cdot P = C_2 - d \cdot C_1$

Modular Addition on FPGA

Method 1 : Computing Mod using repeated subtraction

```
module modular_adder_repeated_subtract #(
   parameter WIDTH = 4  // bit-width (adjust as needed)
)(
   input  wire clk,
   input  wire rst,
   input  wire start,
   input  wire [WIDTH-1:0] a,
   input  wire [WIDTH-1:0] b,
   input  wire [WIDTH-1:0] m,
   output reg  [WIDTH-1:0] result,
   output reg  done
);
   reg [WIDTH:0] sum;   // One bit wider to prevent overflow
   reg busy;

   always @(posedge clk or posedge rst) begin
      if (rst) begin
         result <= 0;
         sum <= 0;
         done <= 0;
         busy <= 0;
      end else begin
         if (start && !busy) begin
            sum <= a + b;
            busy <= 1;
            done <= 0;
         end else if (busy) begin
            if (sum >= m) begin
               sum <= sum - m;
            end else begin
               result <= sum[WIDTH-1:0];
               done <= 1;
               busy <= 0;
            end
         end
      end
   end
endmodule
```

Problem with this method :

Lets take A = 1000000000000 and M = 2

Number of Subtraction = 500000000000

When we take a big number as A and a small number for M it will result in a big number of subtraction operation to determine the A mod M. So this method is not very efficient for such cases

Method 2 : Computing Mod using Long Division

```
module Shift_Registers (
   input wire clk,
   input wire rst,
   input wire start,
   input wire [3:0] r1_in,
   input wire [3:0] r2_in,
   input wire [3:0] m,
   output reg [4:0] out,
   output reg busy
);
   reg [4:0] r3;
   reg [4:0] temp;
   reg [2:0] count;

   always @(posedge clk or posedge rst) begin
      if (rst) begin
         r3 <= 0;
         out <= 0;
         temp <= 0;
         busy <= 0;
         count <= 0;
      end else begin
         if (start && !busy) begin
            r3 <= r1_in + r2_in;
            out <= 5'b00000;
            busy <= 1;
            count <= 0;
         end else if (busy) begin
            temp = {out[3:0], r3[4]};
            if (temp >= m) begin
               temp = temp - m;
            end

            out <= temp;
            r3 <= {r3[3:0], 1'b0};
            count <= count + 1;

            if (count == 4) begin
               busy <= 0;
            end
        ......
```

Number of Subtraction = 12

This is a significant improve from the previous method. We have implemented this method on FPGA

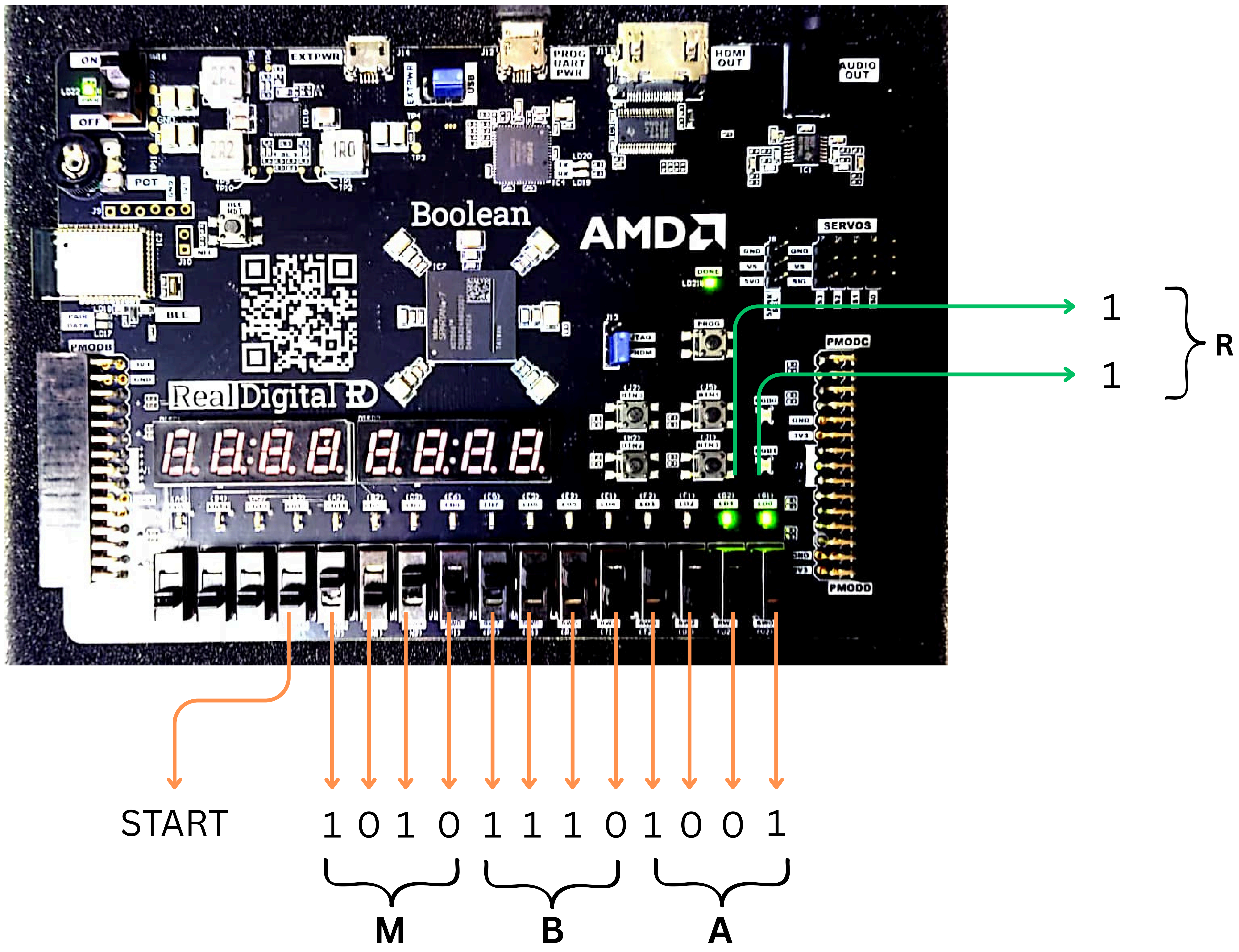
FPGA board: Boolean board
Product family: Spartan-7
Project part: xc7s50csga324-1

A (9) = 1001

B (14) = 1110

M (10) = 1010

R (3) = 0011



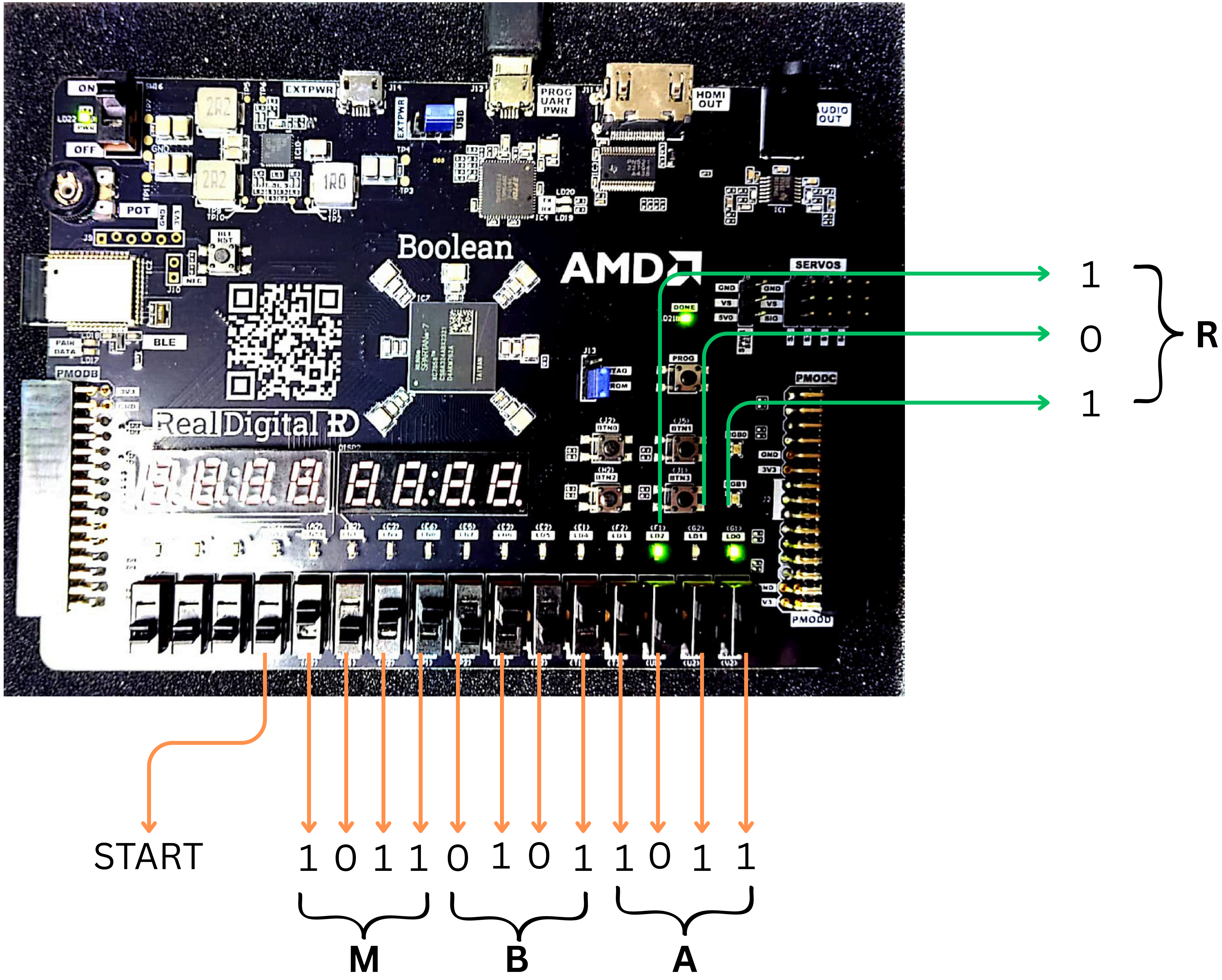
FPGA board: Boolean board
Product family: Spartan-7
Project part: xc7s50csga324-1

A (11) = 1011

B (5) = 0101

M (11) = 1011

R (5) = 0101



Future Plans

- Determining all the operations that will be needed for implementation of the ElGamal algorithm on FPGA.

- Finding the efficient approach for all the operations.

- Finally implementing the complete ElGamal algorithm on FPGA board.

Thank You