HW/SW Co-Design of Elliptic Curve Cryptography on 8051

Term Project Report

Group7

B07901010范詠為、B07901013林子軒

1. ECC Algorithm
   a. Abstract

In the field of Information Security, Cryptography is one of the many ways to secure the insecure information channels. In 1985, Neal Koblitz and Victor Miller independently introduced elliptic curve cryptography.

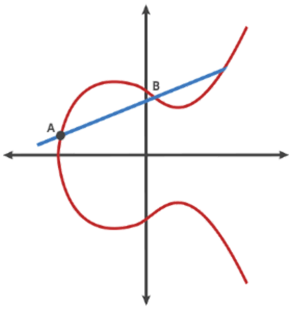
ECC uses the set of points on an elliptic curve along with an addition rule. The unique mathematical structure of the points with the addition rule enables us to perform encryption and decryption of plaintexts. Another reason that supports the feasibility of ECC is the fact that it uses significantly smaller key sizes than the RSA Cryptosystem.

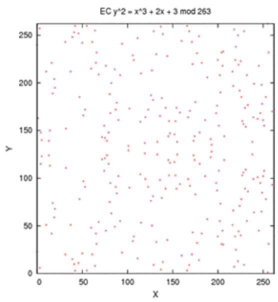
	RSA	ECC
Key Size (security 280)	1024-bits	160-bits
Pros	easy implementation	fast, smaller key size
Cons	slow, longer key size	more complicated

Since the performance of 8-bit microcontrollers is often too poor for the implementation of public-key cryptography in software, we designed a hardware accelerator for ECC on an 8051 microcontroller to perform scalar multiplication over GF(2^8).

   b. Elliptic Curve (weierstrass equation)

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





   c. Finite Field
      i. GF(p), p is prime number
      ii. GF(2^m) (i.e. in binary field, is more preferable on HW)
      iii. GF(p^m)
   d. Field Arithmetic on GF(2^m)

Field Arithmetic on GF(2^m)

- Addition/Subtraction
  - XOR
  - {01010011}+{11001010}={10011001}

$x^6 + x^4 + x + 1$ => 01010011
$x^7 + x^6 + x^3 + 1$ => 11001010

- Multiplication
  - irreducible polynomial $x^8 + x^4 + x^3 + x + 1$=10011011
  - $(x^6 + x^4 + x + 1)(x^7 + x^6 + x^3 + x) = x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$
  - =11111101111110 mod 100011011=00000001
- Multiplicative inversion
  - $a^{-1} = a^{(2^m-2)}$

e. ECC Point Arithmetic

i. Point Negation (PN, -P)

Algorithm 1 Point negation

INPUT: point $P(x_1, y_1)$.
OUTPUT: point $-P$.
$X \leftarrow x_1$
$Y \leftarrow x_1 \oplus y_1$
return (X, Y)

ii. Point Doubling (PD, 2P)

Algorithm 2 Point doubling

INPUT: point $P(x_1, y_1)$.
OUTPUT: point $2P$.
if $P = -P$ or $P = \mathcal{O}$ **then**
  return $\mathcal{O}$
else
  $\lambda \leftarrow x_1 \oplus y_1 / x_1$
  $X \leftarrow \lambda^2 \oplus \lambda \oplus a$
  $Y \leftarrow x_1^2 \oplus \lambda \cdot X \oplus X$
  return (X, Y)
end if

iii. Point Addition (PA, P+Q)

Algorithm 3 Point addition

INPUT: points $P(x_1, y_1)$, $Q(x_2, y_2)$.
OUTPUT: point $P + Q$.
if $P \neq Q$ **then**
  if $P = -Q$ **then**
    return $\mathcal{O}$
  else if $P = \mathcal{O}$ **then**
    return Q
  else if $Q = \mathcal{O}$ **then**
    return P
  else
    $\lambda \leftarrow (y_2 \oplus y_1)/(x_2 \oplus x_1)$
    $X \leftarrow \lambda^2 \oplus \lambda \oplus x_1 \oplus x_2 \oplus a$
    $Y \leftarrow \lambda \cdot (x_1 \oplus X) \oplus X \oplus y_1$
    return (X, Y)
  end if
else
  return $2P$
end if

iv. Scalar Multiple (SM, nP)

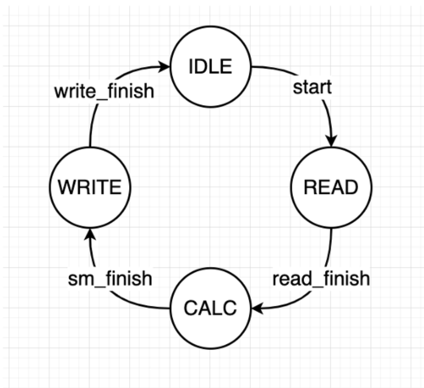
Algorithm 4 Scalar multiple

INPUT: point P, integer n.
OUTPUT: point nP.
$A \leftarrow P$
$R \leftarrow \mathcal{O}$
while $n > 0$ **do**
  if $n \equiv 1 \bmod 2$ **then**
    $R \leftarrow R + A$
  end if
  $n \leftarrow n \gg 1$
  $A \leftarrow 2A$
end while
return R

f. Encryption

Assume Alice has a random integer ka, which is her private key. The public key of Bob is kb*P, where kb is Bob’s private key and P is an initial point on a specific elliptic curve, which is also public. If Alice want to send messages to Bob, she can use ka*(kb*P) and use this to encrypt the messages, and the “key exchange” process is completed.

2. System Architecture
   a. HW
      i. ECC (connected to 8051, perform scalar multiplication)
         1. fsm



      ii. SCALAR_MULT
         1. POINT_ADDITION
         2. POINT_DOUBLING
         3. POINT_NEGATION
      iii. CORE (8 PEs) - 8-bit serial parallel multiplier, can perform add/mul
         1. ADD (~2 cycles)
         2. MULTIPLY (~2 cycles)
         3. INVERSE (~100 cycles)
      iv. Block Diagram of Core Multiplier
          we use similar structure as below but only 8-bit (8-PEs)

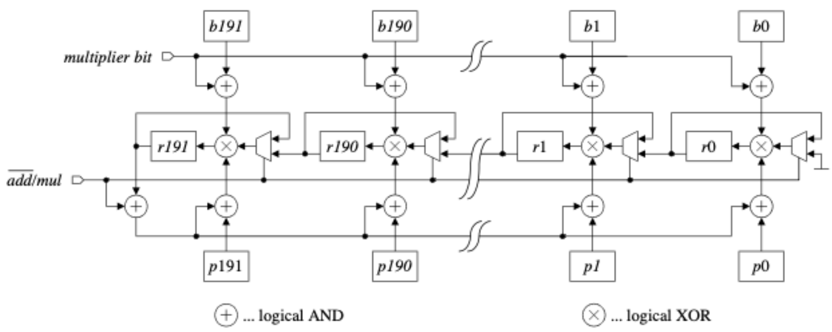


Fig. 3. Datapath of the GF(2^m) arithmetic unit for operands up to 192 bits

   b. SW
      i. 8051(connected to ECC, give data and receive output result)
      ii. IO ports
         1. outputs
            a. clk, rst_n

b. P0 for ECC start signal

c. P1 for ECC input data (n, P, a, p)

2. inputs

a. P2 for ECC output result (P_o)

b. P3 for ECC output valid signal

iii. input data cutting and output data concatenate:

The length of each port is only 8bit, the input data should be split and be sent in multiple cycles. Similarly, the output data should be integrated to get the whole value.

3. Simulation Result (using io_port.t)

a. Terminal

```
fanyungwei@ip87-53 build % ./main ../io_port.t ../null.dmp

        SystemC 2.3.3-Accellera --- Jun 11 2022 02:18:10
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED

time            P0(start)       P1(in_data)     P2(out_data)    P3(valid)

Info: (I702) default timescale unit used for tracing: 1 ps (waveform.vcd)
start input...
2 us            00000001        00000000        00000000        00000000
2010 ns         00000001        00111011        00000000        00000000
2020 ns         00000001        00000000        00000000        00000000
2030 ns         00000001        11100110        00000000        00000000
2040 ns         00000001        11001010        00000000        00000000
2050 ns         00000001        10101010        00000000        00000000
2060 ns         00000001        00011011        00000000        00000000
finish, output...
23100 ns        00000001        00011011        00000000        00000001
23110 ns        00000001        00011011        10100011        00000001
23120 ns        00000001        00011011        11110111        00000001
```

when P0 == 1, start giving input data

n=59, (00111011)

P=59082, (0, 11100110, 11001010)

a= 170, (10101010)

p= 27, (00011011) // (x^8)+x^4+x^3+x+1

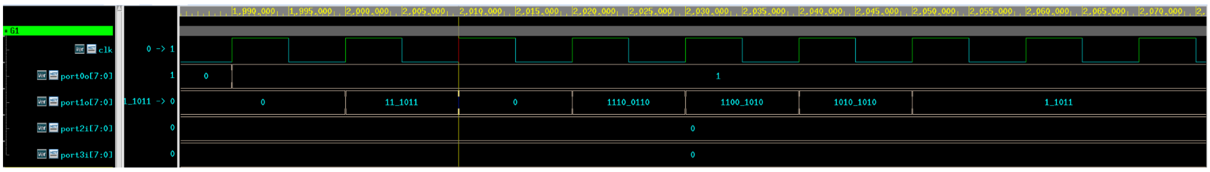
// calculating nP…

when P3 == 1, output is valid

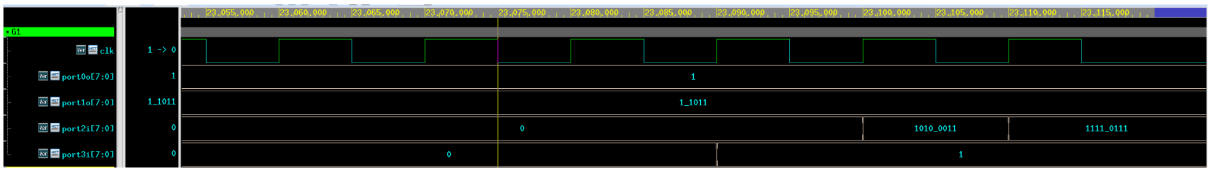
P_o = 41975, (0, 10100011, 11110111)

b. waveform.vcd

i. start, input



ii. finish, output



4. User Manual
    a. Github Link
    https://github.com/BrianEE07/HWSW_Codesign_ECC.git
    b. Program Source List
    _reference/
        contain reference paper
    _slides/
        contain proposal, final slide, final report
    src/
        sw_sim/
            contain software simulation code
        pattern/
            contain test pattern
        8051/CMakeList.txt
            generate Makefile
        8051/io_port.t
            8051 dump input data (generated from io_port.c)
        8051/ecc.h
            main design

    c. How to run (tested on MacOS)
        i. download source code from github
        https://github.com/BrianEE07/HWSW_Codesign_ECC.git
        ii. run software simulation
            1. cd HWSW_Codesign_ECC/src/sw_sim
            2. g++ ecc.cpp -o ecc.o
            3. ./ecc.o
        iii. run HWSW codesign (systemc on 8051)
            1. cd HWSW_Codesign_ECC/src/8051/build
            2. mkdir build
            3. cd build
            4. cmake ../
            5. make
            6. ./main ../io_port.t ../null_dump
        result will be shown in terminal, and the waveform (.vcd) will be dumped in the current directory.

5. References

[1] Koschuch, M. et al. (2006). Hardware/Software Co-design of Elliptic Curve Cryptography on an 8051 Microcontroller. In: Goubin, L., Matsui, M. (eds) Cryptographic Hardware and Embedded Systems - CHES 2006. CHES 2006. Lecture Notes in Computer Science, vol 4249. Springer, Berlin, Heidelberg.

https://doi.org/10.1007/11894063_34
[2] Implementation of Elliptic Curve Cryptography in Binary Field, D R Susantio and I Muchtadi-Alamsyah 2016 J. Phys.: Conf. Ser. 710 012022