Elliptic Curve Cryptosystems

Çetin Kaya Koç

Oregon State University

Elliptic Curve Cryptosystems

Elliptic curves defined over $GF(p)$ or $GF(2^k)$ are used in cryptography

The arithmetic of $GF(p)$ is the usual mod p arithmetic

The arithmetic of $GF(2^k)$ is similar to that of $GF(p)$, however, there are some differences

Elliptic curves over $GF(2^k)$ are more popular due to the space and time-efficient algorithms for doing arithmetic in $GF(2^k)$

Elliptic curve cryptosystems based on discrete logarithms seem to provide similar amount of security to that of RSA, but with relatively shorter key sizes

Elliptic Curves over $GF(p)$

Let $p > 3$ be a prime number and $a, b \in GF(p)$ be such that $4a^3 + 27b^2 \neq 0$ in $GF(p)$. An elliptic curve E over $GF(p)$ is defined by the parameters a and b as the set of solutions (x, y) where $x, y \in GF(p)$ to the equation

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

together with an extra point O. The set of points E form a group with respect to the addition rules:

- $O + O = O$
- $(x, y) + O = (x, y)$
- $(x, y) + (x, -y) = O$

Elliptic Curves over $GF(p)$

- Addition of two points with $x_1 \neq x_2$

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

- Doubling of a point with $x_1 \neq 0$

$$(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$$

$$\begin{aligned} \lambda &= (3x_1^2 + a)(2y_1)^{-1} \\ x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

Example: Let the elliptic curve be defined as the solutions of

$$y^2 = x^3 + x + 1$$

over the field $GF(23)$

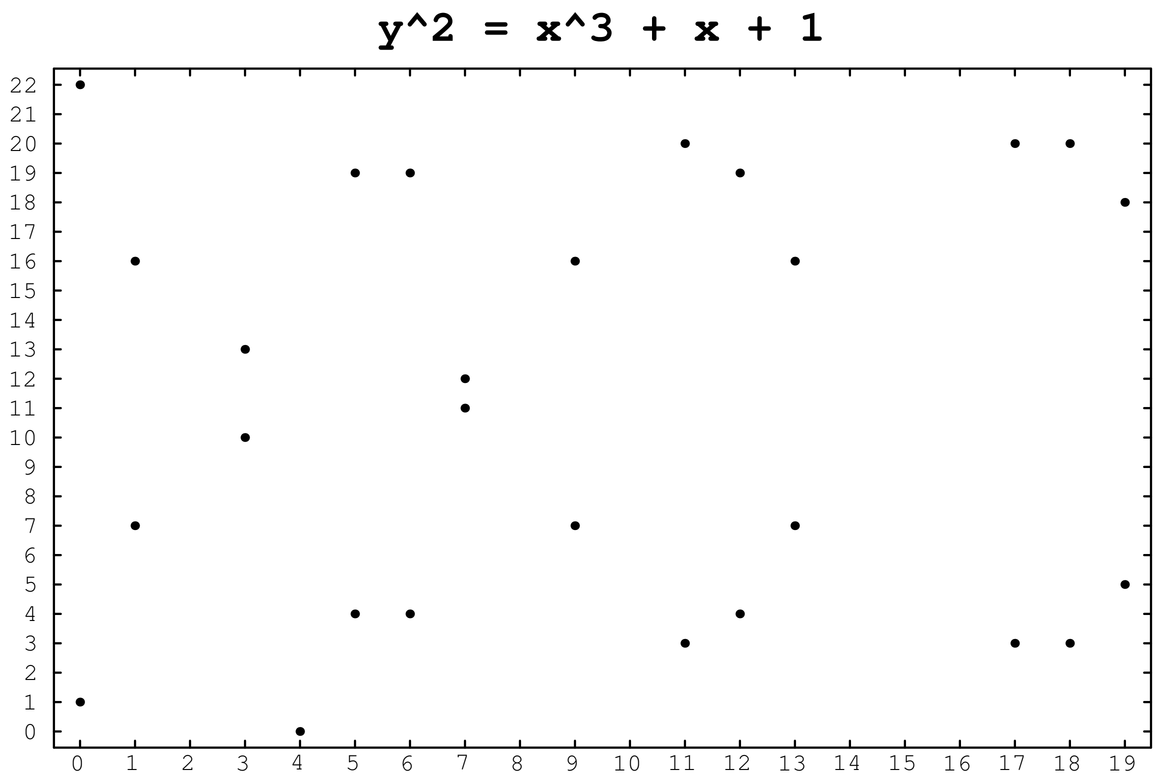
The group E has 28 points including $\mathbf{O}$

Addition: $(3, 10) + (9, 7) = (17, 20)$

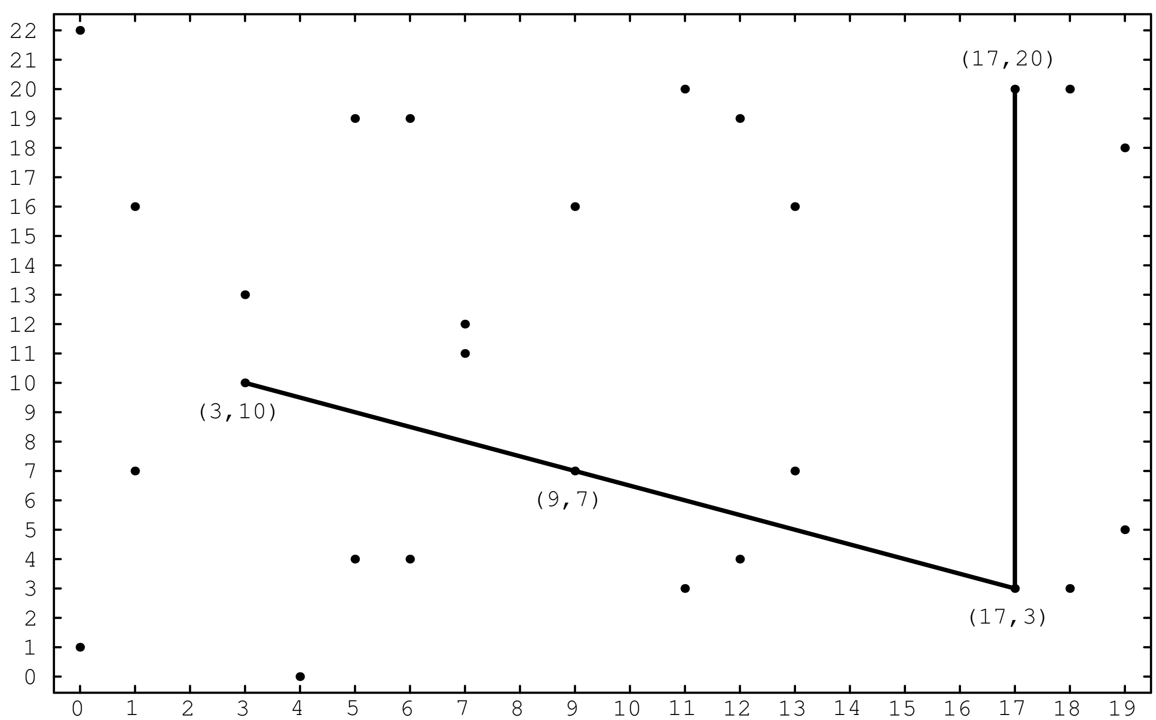
$$\begin{aligned} \lambda &= (7 - 10)(9 - 3)^{-1} = (-3)(6)^{-1} = 11 \\ x_3 &= 11^2 - 3 - 9 \;=\; 17 \\ y_3 &= 11(3 - 17) - 10 \;=\; 20 \end{aligned}$$

Doubling: $(3, 10) + (3, 10) = (7, 12)$

$$\begin{aligned} \lambda &= (3(3^2) + 1)(20)^{-1} \;=\; 6 \\ x_3 &= 6^2 - 6 \;=\; 7 \\ y_3 &= 6(3 - 7) - 10 \;=\; 12 \end{aligned}$$

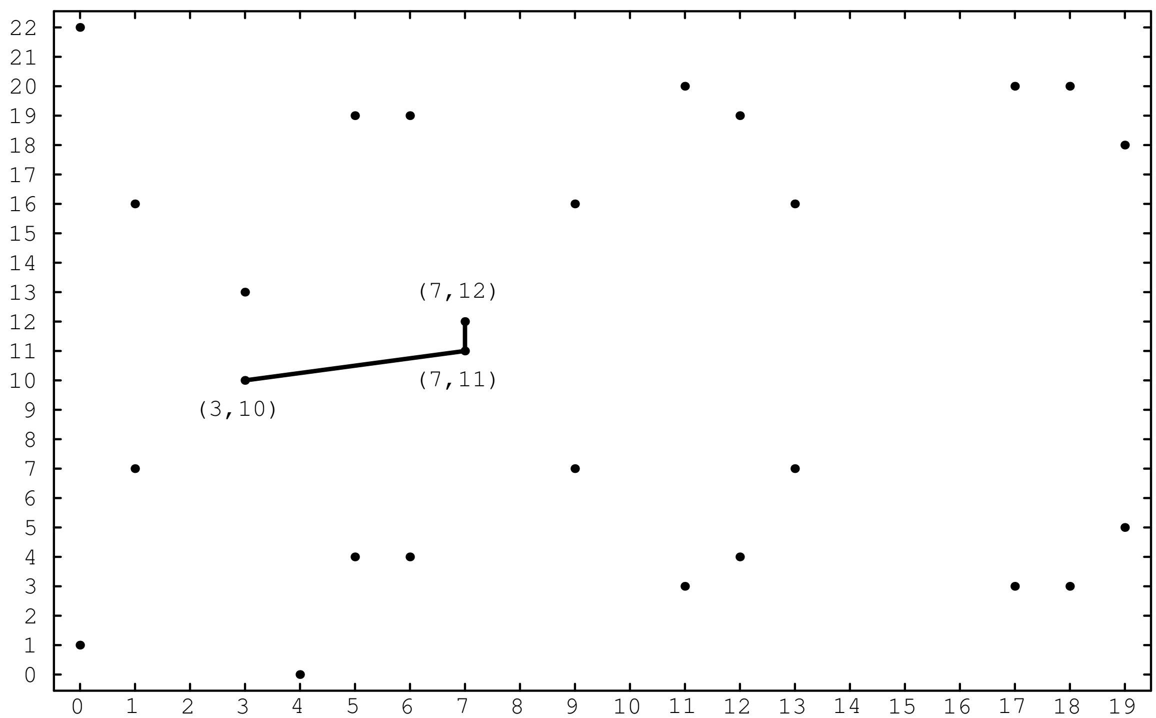

y^2 = x^3 + x + 1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

(3,10) + (9,7) = (17,20)



(17,20)
(3,10)
(9,7)
(17,3)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

(3,10) + (3,10) = (7,12)



Elliptic Curves over $GF(2^k)$

A non-supersingular elliptic curve E over the field $GF(2^k)$ is defined by parameters $a, b \in GF(2^k)$ with $b \neq 0$ is the set of solutions (x, y) where $x, y \in GF(2^k)$, to the equation

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

together with an extra point O. The set of points E form a group with respect to the addition rules:

- $O + O = O$
- $(x, y) + O = (x, y)$
- $(x, y) + (x, x + y) = O$

Elliptic Curves over $GF(2^k)$

- Addition of two points with $x_1 \neq x_2$

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

$$\begin{aligned} \lambda &= (y_1 + y_2)(x_1 + x_2)^{-1} \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \end{aligned}$$

- Doubling of a point with $x_1 \neq 0$

$$(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$$

$$\begin{aligned} \lambda &= x_1 + (y_1)(x_1)^{-1} \\ x_3 &= \lambda^2 + \lambda + a \\ y_3 &= x_1^2 + (\lambda + 1)x_3 \end{aligned}$$

Elliptic Curve Cryptosystems

Based on the difficulty of computing e given eP where P is a point on the curve

Example: Elliptic Curve Diffie-Hellman

Alice and Bob agree on, the elliptic curve E, the underlying field $GF(2^k)$ or $GF(p)$, and the generating point P with order n

- Alice sends $Q = aP$ to Bob
- Bob sends $R = bP$ to Alice
- Alice computes $S = a(R) = abP$
- Bob computes $S = b(Q) = abP$

Adversary knows P, and sees Q and R

Computing S seems to require elliptic logarithms

Elliptic Curve Arithmetic

Computation of eP can be performed using exponentiation algorithms

In order to compute e multiple of P we perform elliptic curve additions

An elliptic curve addition is performed by using a few **finite field** operations

Implementation of elliptic curve addition operation requires implementation of four basic finite field operations: addition, subtraction, multiplication, and inversion

For example, addition of two distinct points requires 2 field multiplications and 1 field inversion

Inversion is a relatively expensive operation

Projective Coordinates

Projective coordinates eliminate the need for performing inversion

In projective coordinates, a point on E has 3 coordinate values

$$(x_1 \ : \ y_1 \ : \ z_1)$$

while the affine coordinates requires only two values: (x_1, y_1)

Given the distinct points P and Q expressed in projective coordinates

$$\begin{aligned} P &= (x_1 \ : \ y_1 \ : \ z_1) \\ Q &= (x_2 \ : \ y_2 \ : \ z_2) \end{aligned}$$

We compute the projective coordinates of the elliptic sum

$$P + Q \ = \ (x_3 \ : \ y_3 \ : \ z_3)$$

Projective Coordinates

The projective addition formulae

$$\begin{aligned} A &= x_2 z_1 + x_1 \\ B &= y_2 z_1 + y_1 \\ C &= A + B \\ D &= A^2(A + a z_1) + z_1 BC \end{aligned}$$

$$\begin{aligned} x_3 &= AD \\ y_3 &= CD + A^2(Bx_1 + Ay_1) \\ z_3 &= A^3 z_1 \end{aligned}$$

This computation requires 13 field multiplications, and no inversion

Projective Coordinates

Similarly, the addition formulae for computing $2P$ is given as

$$\begin{aligned} A &= x_1 z_1 \\ B &= b z_1^4 + x_1^4 \end{aligned}$$

$$\begin{aligned} x_3 &= AB \\ y_3 &= x_1^4 A + B(x_1^2 + y_1 z_1 + A) \\ z_3 &= A^3 \end{aligned}$$

This computation requires 7 field multiplications, and no inversion

Thus, we have eliminated the inversions at the expense of

- storing 3 $GF(2^k)$ values to represent P
- performing a few more multiplications

Exponentiation Heuristics

Given the integer e, the computation of eP is an exponentiation operation

The objective is to use as few elliptic curve additions as possible for a given integer e

This problem is related to **addition chains**

An addition chain is a sequence of integers

$$a_0 \quad a_1 \quad a_2 \quad \cdots \quad a_r$$

starting from $a_0 = 1$ and ending with $a_r = e$ such that any a_k is the sum of two earlier integers a_i and a_j in the chain:

$$a_k = a_i + a_j \quad \text{for } 0 < i, j < k$$

Addition Chains

Example: $e = 55$

1	2	3	6	12	13	26	27	54	55
1	2	3	6	12	13	26	52	55	
1	2	4	5	10	20	40	50	55	
1	2	3	5	10	11	22	44	55	

An addition chain yields an algorithm for computing eP given the integer e

$$P \quad 2P \quad 3P \quad 5P \quad 10P \quad 11P \quad 22P \quad 44P \quad 55P$$

The length of the chain r gives the number of operations required to compute eP

Addition Chains

Finding the shortest addition chain is an NP-complete problem

Let $H(e)$ be the Hamming weight of e

Upper bound: $\lfloor \log_2 e \rfloor + H(e) - 1$

Lower bound: $\log_2 e + \log_2 H(e) - 2.13$

Heuristics: binary, m-ary, sliding windows

Statistical methods, such as simulated annealing, can be used to produce short addition chains for certain exponents

Binary Method

Scan the bits of e and perform elliptic curve doublings and additions in order to compute $Q = eP$

1. **if** $e_{k-1} = 1$ **then** $Q := P$ **else** $Q := O$
2. **for** $i = k - 2$ **downto** 0

   2a. $Q := Q + Q$

   2b. **if** $e_i = 1$ **then** $Q := Q + P$
3. **return** Q

Example: $e = 55 = (110111)$

Step 1: $e_5 = 1 \longrightarrow Q := P$

i	e_i	Step 2a (Q)	Step 2b (Q)
4	1	$P + P = 2P$	$2P + P = 3P$
3	0	$3P + 3P = 6P$	$6P$
2	1	$6P + 6P = 12P$	$12P + P = 13P$
1	1	$13P + 13P = 26P$	$26P + P = 27P$
0	1	$27P + 27P = 54P$	$54P + P = 55P$

Addition-Subtraction Chains

An addition-subtraction chain is a sequence of integers

$$a_0 \quad a_1 \quad a_2 \quad \cdots \quad a_r$$

starting from $a_0 = \pm 1$ and ending with $a_r = e$ such that any a_k is the sum or the difference of two earlier integers a_i and a_j in the chain:

$$a_k = a_i \pm a_j \quad \text{for } 0 < i, j < k$$

Example: $e = 55$

$$\pm 1 \quad 2 \quad 4 \quad 8 \quad 7 \quad 14 \quad 28 \quad 56 \quad 55$$

An addition-subtraction chain is an algorithm for computing eP given the integer e

However, it requires negative multiples of P

Signed-Digit Recoding

A signed-digit recoding of e is a representation of the integer e using the digits $\{-1, 1, 0\}$

Once a signed-digit recoding of e is obtained, it can be scanned digit-by-digit in a way similar to the binary method:

- No elliptic curve addition if $e_i = 0$
- An elliptic curve addition using P if $e_i = 1$
- An elliptic curve addition using $-P$ if $e_i = -1$

Signed-Digit Recoding Binary Method

Addition-subtraction chains are suitable for elliptic curves since computing $-P$ is trivial

For elliptic curves over $GF(p)$:
if $P = (x, y)$, then $-P = (x, -y)$

Non-supersingular elliptic curves over $GF(2^k)$:
if $P = (x, y)$, then $-P = (x, x + y)$

Input: $P, -P, e$
Output: $Q := eP$

0. Obtain a signed-digit recoding f of e
1. **if** $f_k = 1$ **then** $Q := P$ **else** $Q := O$
2. **for** $i = k - 1$ **downto** 0
2a. $Q := Q + Q$
2b. **if** $f_i = 1$ **then** $Q := Q + P$
if $f_i = \bar{1}$ **then** $Q := Q + (-P)$
3. **return** Q

Canonical Recoding Algorithm

This algorithm optimally encodes the exponent using the digits $\{0, 1, \bar{1}\}$

e_{i+1}	e_i	a_i	f_i	a_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	$\bar{1}$	1
1	1	0	$\bar{1}$	1
1	1	1	0	1

For example, $e = 3038$ is encoded as

$$\begin{aligned} e &= (0101111011110) \\ f &= (10\bar{1}0000\bar{1}000\bar{1}0) \end{aligned}$$

requiring 3 elliptic curve additions instead of 9 (in addition to the elliptic curve doublings)

Properties of $GF(2^k)$ Arithmetic

An element a of $GF(2^k)$ is usually represented as a binary vector $(a_{k-1}a_{k-2}\cdots a_1a_0)$

- The terms a_i may interpreted as the coefficients of the polynomial

$$a_{k-1}x^{k-1} + a_{k-1}x^{k-1} + \cdots + a_1x + a_0$$

- The elements of $GF(2^k)$ can be viewed as a vector space of dimension k over $GF(2)$. In this case, there exists a set of k elements (called the basis)

$$\alpha_0, \alpha_1, \ldots, \alpha_{k-1} \in GF(2^k)$$

  such that a can be written uniquely in the form

$$a = a_0\alpha_0 + a_1\alpha_1 + \cdots + a_{k-1}\alpha_{k-1}$$

Addition in $GF(2^k)$

An element A of $GF(2^k)$ is represented using either the polynomial basis

$$A = (A_{k-1}A_{k-2}\cdots A_1A_0) = \sum_{i=0}^{k-1} A_i x^i$$

or the vector space basis

$$A = (A_{k-1}A_{k-2}\cdots A_1A_0) = \sum_{i=0}^{k-1} A_i \alpha^i$$

where $\alpha_i \in GF(2^k)$ are known in advance

In either case, the computation of

$$C = (C_{k-1}C_{k-2}\cdots C_1C_0) = A + B$$

is easily performed by component-wise modulo 2 addition (the XOR operation)

$$\begin{aligned} C_i &= A_i + B_i \pmod 2 \\ &= A_i \oplus B_i \end{aligned}$$

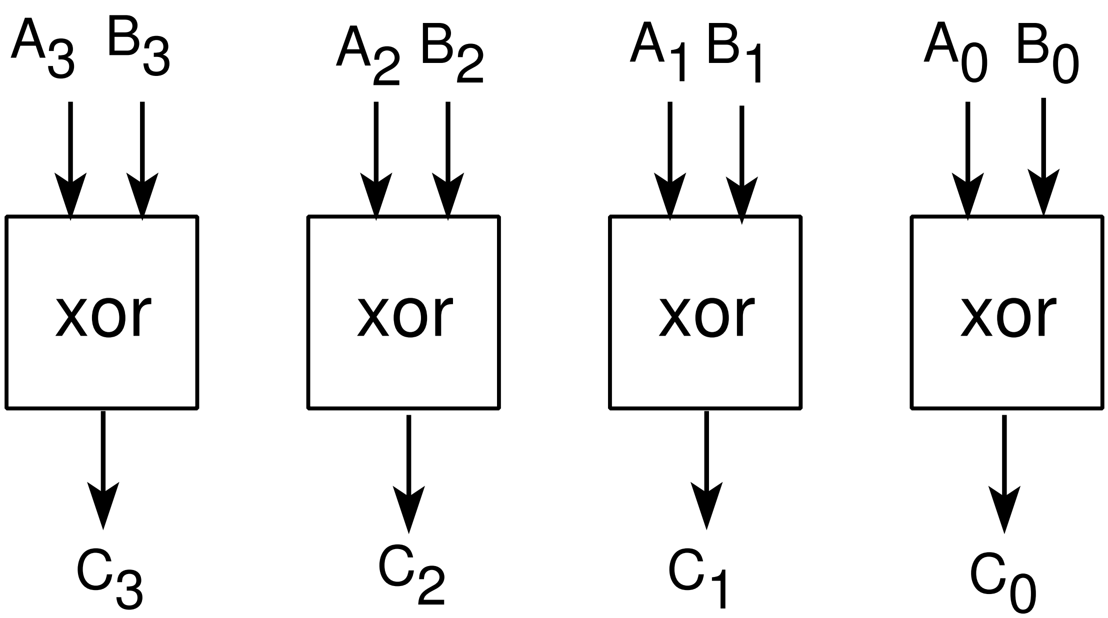
for $i = 0, 1, \ldots, k-1$

- The total delay is $O(1)$ (single XOR delay)

- The total area is $k \times$ XOR area

- Scales up easily for large k

- Subtraction is easy: The same as addition



Multiplication in $GF(2^k)$

<u>Using polynomial basis:</u> We find an irreducible polynomial of degree k

$$f(x) = x^k + f_{k-1}x^{k-1} + \cdots + f_1 x + f_0$$

The multiplication of $C = A \cdot B$ in $GF(2^k)$ is performed by multiplying the polynomials $A(x)$ and $B(x)$ modulo $f(x)$

This is similar to Multiply and Reduce method of modular multiplication. Multiplication algorithms (such as interleaving) can be used

<u>Using vector space basis:</u> Squaring and multiplication operations can be significantly simplified by judicious selection of the basis

For example, a normal basis can be used

Squaring in a Normal Basis

A normal basis of $GF(2^k)$ is a basis of the form

$$\{\beta, \beta^2, \beta^4, \ldots, \beta^{2^{k-1}}\}$$

where β is an element of $GF(2^k)$. It is well-known that such a basis always exists. Let A be expressed in a normal basis. We have

$$\begin{aligned} A &= (a_{k-1}a_{k-2}\cdots a_1 a_0) \\ &= a_0\beta + a_1\beta^2 + a_2\beta^4 + \cdots + a_{k-1}\beta^{2^{k-1}} \end{aligned}$$

We compute the square of A as

$$\begin{aligned} A^2 &= (\sum_{i=0}^{k-1} a_i\beta^{2^i}) \cdot (\sum_{i=0}^{k-1} a_i\beta^{2^i}) \\ &= \sum_{i=0}^{k-1} \left(a_i\beta^{2^i}\right)^2 = \sum_{i=0}^{k-1} a_i\beta^{2^{i+1}} \\ &= (a_{k-2}a_{k-3}\cdots a_1 a_0 a_{k-1}) \end{aligned}$$

which is a cyclic left shift of A

Multiplication in a Normal Basis

The product $C = AB$ is given as

$$C = \sum_{i=0}^{k-1} C_i \beta^{2^i} = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} A_i B_j \beta^{2^i + 2^j}$$

Since $\beta^{2^i+2^j}$ is also an element of $GF(2^k)$, it can be expressed as

$$\beta^{2^i+2^j} = \sum_{r=0}^{k-1} \lambda_{ij}^{(r)} \beta^{2^r}$$

where $\lambda_{ij}^{(r)} \in GF(2)$. This yields a formulae

$$C_r = \sum_{i=0}^{k-1} A_i B_i \lambda_{ij}^{(r)} \qquad \text{for } 0 \le r \le k-1$$

We also notice that

$$\beta^{2^{i-s}+2^{j-s}} = \sum_{r=0}^{k-1} \lambda_{i-s,j-s}^{(r)} \beta^{2^r} = \sum_{r=0}^{k-1} \lambda_{ij}^{(r)} \beta^{2^{r-s}}$$

which implies

$$\lambda_{ij}^{(s)} = \lambda_{i-s,j-s}^{(0)} \qquad \text{for all } 0 \le i, j, s \le k-1$$

Thus, we have a formula for C_r as

$$C_r = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} A_{i+r} B_{j+r} \lambda_{ij}$$

This formulae has remarkable properties:

- Consider a circuit built for computing C_0 which receives the inputs as (in this order)

$$A_0, A_1, \ldots, A_{k-2}, A_{k-1}$$
$$B_0, B_1, \ldots, B_{k-2}, B_{k-1}$$

uses the formulae to compute

$$C_0 = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} A_i B_j \lambda_{ij}$$

The same circuit can be used to compute C_1 with the inputs as

$$A_1, A_2, \ldots, A_{k-1} A_0$$
$$B_1, B_2, \ldots, B_{k-1} B_0$$

• The number of nonzero λ_{ij}s determine the complexity of the multiplication circuit

The upper-bound is k^2

The lower-bound is shown to be $2k-1$

A normal basis with $2k-1$ nonzero λs is called an <u>optimal normal basis</u>

Such basis exists for certain fields

• Thus, a circuit with area $O(k)$ can be built to multiply two elements of $GF(2^k)$ in k clock cycles

Inversion in $GF(2^k)$

An efficient algorithm for computing an inverse of an element of $GF(2^k)$ was proposed by Itoh, Teechai, and Tsujii

If $a \in GF(2^k)$ and $a \neq 0$, then

$$a^{-1} = a^{2^k - 2} = \left(a^{2^{k-1}-1}\right)^2$$

For k even or odd, we have
Odd:

$$2^{k-1} - 1 = (2^{(k-1)/2} - 1) \cdot (2^{(k-1)/2} + 1)$$

Even:

$$2^{k-1} - 1 = 2 \cdot (2^{(k-2)/2} - 1) \cdot (2^{(k-2)/2} + 1)$$

These formulae yield an algorithm for computing the inverse by using factorization of the exponent

Example of Inverse Computation

Consider the field $GF(2^{155})$

$$\begin{aligned}
2^{155} - 2 &= 2 \cdot (2^{77} - 1) \cdot (2^{77} + 1) \\
2^{77} - 1 &= 2 \cdot (2^{38} - 1) \cdot (2^{38} + 1) + 1 \\
2^{38} - 1 &= (2^{19} - 1) \cdot (2^{19} + 1) \\
2^{19} - 1 &= 2 \cdot (2^{9} - 1) \cdot (2^{9} + 1) + 1 \\
2^{9} - 1 &= 2 \cdot (2^{4} - 1) \cdot (2^{4} + 1) + 1 \\
2^{4} - 1 &= (2^{2} - 1) \cdot (2^{2} + 1) \\
2^{2} - 1 &= (2^{1} - 1) \cdot (2^{1} + 1)
\end{aligned}$$

It requires 10 multiplications to compute an inverse in $GF(2^{155})$

In general, the method requires

$$\lfloor \log_2(k-1) \rfloor + H(k-1) - 1$$

field multiplications

Implementation Results

<u>Elliptic Curves</u>

Newbridge Microsystems (1988)

- Uses the field $GF(2^{593})$
- Clockrate 20 MHz
- Field Multiplication: 65 μs
- Inversion: 2.5 ms

Agnew, Mullin, Vanstone (1993)

- Uses the field $GF(2^{155})$
- Clockrate 40 MHz
- Field Multiplication: 4 μs
- Inversion: 95 μs

Software Implementation of ElGamal

- Uses the field $GF(2^{104})$
- Sun-2 Sparcstation
- 105-bit Encryption: 500 msec*