Scalar Multiplication and Addition Chains

Peter Birkner

Department of Mathematics, Technical University of Denmark

Summer School on Elliptic and Hyperelliptic Curve Cryptography, Toronto 2006

Outline

Motivation

Given: A group $(G, \oplus)$, an element $P \in G$ and a scalar $n \in \mathbb{Z}$

Task: Compute $[n]P$ efficiently

- In Elliptic curve cryptosystems G is group of points on the curve.
- Scalar multiplication is the most important operation in these DL-based cryptosystems!
- First naive method: $[n]P = P \oplus P \oplus \cdots \oplus P$ (n-times)
- If $n = 2^k$, then compute $[n]P$ using k doublings $[2]P$, $[4]P$, $[8]P$, $\ldots$, $[2^k]P$

Better: Left-To-Right Binary (1)

Algorithm 1 (Left–to–right binary)

IN: An element $P \in G$ and a positive integer $n = (n_{l-1} \dots n_0), n_{l-1} = 1$.

OUT: The element $[n]P \in G$.

1. $R \leftarrow P$
2. `for` $i = l - 2$ `to` 0 `do`
   1. $R \leftarrow [2]R$
   2. `if` $n_i = 1$ `then` $R \leftarrow R \oplus P$
   3. $i \leftarrow i - 1$
3. `return` R

Left-To-Right Binary (2)

The algorithm uses the following rule:

$$[(n_{l-1} \dots n_i)_2]P = [2]([(n_{l-1} \dots n_{i+1})_2]P) \oplus [n_i]P$$

Example: $45 = (101101)_2$

P
$2P$
$2(2P) \oplus P$
$2(2(2P) \oplus P) \oplus P$
$2(2(2(2P) \oplus P) \oplus P)$
$2(2(2(2(2P) \oplus P) \oplus P)) \oplus P \quad = [45]P$

Algorithm is aka Double-and-Add

Right-To-Left Binary

Algorithm 2 (Right–to–Left binary)

IN: An element $P \in G$ and a positive integer $n = (n_{l-1} \dots n_0), n_{l-1} = 1$.

OUT: The element $[n]P \in G$.

1. $R \leftarrow 0,\ S \leftarrow P,\ i \leftarrow 0$
2. `while` $i \leq l-1$ `do`
   1. `if` $n_i = 1$ `then` $R \leftarrow R \oplus S$
   2. $S \leftarrow [2]S$
   3. $i \leftarrow i+1$
3. `return` R

Remarks

- Right-to-left binary needs $l - 1$ doublings and $w(n)$ additions
- $w(n)$ denotes the Hamming weight of n. That is the number of nonzero digits in the binary representation of n
- On average the density is $1/2$.

Non-Adjacent-Form (NAF) (1)

- On an EC addition and subtraction can be computed with the same effort
- Hence, use **signed** digits!
- $n = \sum_{i=0}^{l-1} n_i 2^i$ with $n_i \in \{0, \pm 1\}$
- No two consecutive digits are nonzero in NAF
- NAF is unique and has minimal density of all signed digit representations
- The average density is $1/3$
- Note: The length can increase by 1

Non-Adjacent-Form (NAF) (2)

Algorithm 3 (Signed-binary representation in NAF)

IN: A positive integer $n = (n_l n_{l-1} \dots n_0)_2$ with $n_l = n_{l-1} = 0$.
OUT: The signed-binary representation of n in NAF $(n'_{l-1} \dots n'_0)_s$.

1. $c_0 \leftarrow 0$
2. `for` $i = 0$ `to` $\ell - 1$ `do`
   1. $c_{i+1} \leftarrow \lfloor (c_i + n_i + n_{i+1})/2 \rfloor$
   2. $n'_i \leftarrow c_i + n_i - 2c_{i+1}$
3. `return` $(n'_{\ell-1} \dots n'_0)_s$

Non-Adjacent-Form (NAF) (3)

Example. We want to compute the NAF of $15 = (1111)_2$

i	c_i	c_{i+1}	n_i	n_{i+1}	n'_i
0	0	1	1	1	-1
1	1	1	1	1	0
2	1	1	1	1	0
3	1	1	1	0	0
4	1	0	0		1

The NAF of 15 is $(1,0,0,0,-1)_{NAF}$ with density $2/5$

$15 = (1,0,-1,1,1)$. Signed digit represent. is **not unique!**

Non-Adjacent-Form (NAF) (4)

Algorithm 4 (Left–to–right NAF)

IN: An element $P \in G$ and a positive integer $n = (n_{l-1} \dots n_0), n_{l-1} = 1$.

OUT: The element $[n]P \in G$.

1. $R \leftarrow P$
2. `for` $i = l - 2$ `to` 0 `do`
    1. $R \leftarrow [2]R$
    2. `if` $n_i = 1$ `then` $R \leftarrow R \oplus P$
    3. `if` $n_i = -1$ `then` $R \leftarrow R \oplus (-P)$
    4. $i \leftarrow i - 1$
3. `return` R

The 2^k-ary Method (1)

- Use a larger basis to get sparse representations of n
- A common choice is 2^k as basis
- $S = \{0, 1, \ldots, 2^k - 1\}$ are the digits
- To perform scalar multiplication, first precompute $[s]P$ for all $s \in S$ and use a modified version of Algorithm 1

Example $k = 3$, $S = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$n = 241 = (11|110|001)_2 = (361)_{2^3}$

The 2^k-ary Method (2)

Algorithm 5 (Left–to–right 2^k-ary)

IN: An element $P \in G$ and a positive integer n in 2^k-ary representation $n = (n_{l-1} \ldots n_0)_{2^k}$
Precomputed values $P, [2]P, \cdots, [2^k - 1]P$

OUT: The element $[n]P \in G$.

1. $R \leftarrow [n_{l-1}]P$
2. `for` $i = l - 2$ `to` 0 `do`
    1. $R \leftarrow [2^k]R$
    2. `if` $n_i \neq 0$ `then` $R \leftarrow R \oplus [n_i]P$
    3. $i \leftarrow i - 1$
3. `return` R

The 2^k-ary Method (3)

Example $k = 3$, $S = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$n = 241 = (361)_{2^3}$

Precompute the values $P, [2]P, \ldots, [7]P$

$R = 3P$

$R = 8R = 24P$
$R = R \oplus 6P = 30P$

$R = 8R = 240P$
$R = R \oplus 1P = 241P$

Sliding Window Methods

- To reduce the number of precomputations **sliding** window methods can be used!
- Digits are only the **odd** integers smaller than 2^k and 0
- $S' = \{0, 1, 3, 5, \dots, 2^k - 1\}$
- Consecutive zeros are skipped
- Scan from right to left $\Rightarrow$ block is odd
- **Example** $(k = 3)$

  $$241 = (\underline{1}\ \underline{111}\ 000\ \underline{1})_2$$

- Sliding window is also possible with signed digits!

Multiexponentiation (1)

- Sometimes one needs to compute more than one scalar multiplication and later add the results
- E. g. in checking a signature
- Use a trick to combine doublings
- **Example.** We want to compute $[27]P_0 \oplus [30]P_1$
  $27 = (11011)_2$
  $30 = (11110)_2$
- Scan the columns from left to right and double-and-add:
  $P_0 \oplus P_1$
  $[2](P_0 \oplus P_1) \oplus P_0 \oplus P_1$
  $[2]([2](P_0 \oplus P_1) \oplus P_0 \oplus P_1) \oplus P_1$
  $\ldots = [27]P_0 \oplus [30]P_1$

Multiexponentiation (2)

Remarks

- Some doublings and additions can be saved if $P_0 \oplus P_1$ is precomputed
- Density is $3/4$
- Using NAF instead of binary reduces density to $5/9$
  $P_0 \oplus P_1$ and $P_0 \oplus (-P_1)$ have to be precomputed
- With the Joint Sparse Form (JSF) a density of $1/2$ can be achieved (see Solinas, 2001)