DIAMANT-Summer School on Elliptic and Hyperelliptic Curve Cryptography

Scalar Multiplication

Christophe Doche

Macquarie University, Sydney

http://www.ics.mq.edu.au/~doche/

Overview

Background and Elementary Methods

Window Methods

Signed-Digit Methods

Multi-Scalar Multiplication

Double-Base Representations

Montgomery's Method

Koblitz curves

Background and Elementary Methods

Definition. Given an integer n and a point P on a curve, a scalar multiplication consists in computing

$$[n]P = \underbrace{P + \cdots + P}_{n \text{ times}}$$

This operation is very similar to an exponentation

Only the notation is different (additive instead of multiplicative)

Background and Elementary Methods

The standard way to compute $[n]P$ is the double-and-add method

Remarks.

This method processes the bits of n from left-to right

It relies on the following basic operations:

- addition $P + Q$, when $P \neq \pm -Q$
- doubling $[2]P$

Background and Elementary Methods

We won't discuss the complexities of doublings and additions

There is a vast litterature on the subject

They vary from one representation to another (Weierstraß, Montgomery, Jacobi, Edwards) and from one coordinate systems to another (affine, projective)

Background and Elementary Methods

Algorithm. Double-and-add

INPUT: $P \in E$ and $n = (n_{\ell-1} \ldots n_0)_2$
OUTPUT: $[n]P$

1. $R \leftarrow P_\infty$
2. **for** $i = \ell - 1$ **downto** 0 **do**
3. $\quad R \leftarrow [2]R$
4. $\quad$ **if** $n_i = 1$ **then** $R \leftarrow R + P$
5. **return** R

Background and Elementary Methods

Example. To compute $[13]P$, start from the binary expansion of 13, i.e. $(1101)_2$

The steps in the computations of $[13]P$ are

$$[13]P = [2]\big([2]([2]([2]P_\infty + P) + P)\big) + P$$

There is also a right-to-left approach

Background and Elementary Methods

Algorithm. Right-to-left scalar multiplication

INPUT: $P \in E$ and $n = (n_{\ell-1} \ldots n_0)_2$
OUTPUT: $[n]P$

1. $R \leftarrow P_\infty$
2. **for** $i = 0$ **to** $\ell - 1$ **do**
3. **if** $n_i = 1$ **then** $R \leftarrow R + P$
4. $P \leftarrow [2]P$
5. **return** R

Example. Take $n = 13$ and let us compute $[n]P$

The binary expansion of n is $(1101)_2$

$$[13]P = P + [4]P + [8]P$$

Background and Elementary Methods

The number of nonzero terms divided by the length of the expansion is called the density

For instance, the average density of the binary expansion is $\frac{1}{2}$

Remark. The complexity of both methods is the same

- $\log_2 n$ doublings
- $\frac{1}{2} \log_2 n$ additions on average

Window Methods

The double-and-add method is closely linked to the binary representation of n

However, it can be easily generalized to any basis

In particular, for the basis 2^k where k is a fixed parameter

$$n = (n_{\ell-1} \ldots n_0)_{2^k}$$

there is the left-to-right 2^k-ary method

Window Methods

Algorithm. Left-to-right 2^k-ary method

INPUT: $P \in E$ and $n = (n_{\ell-1} \ldots n_0)_{2^k}$
OUTPUT: $[n]P$

1. $R \leftarrow P_\infty$
2. **for** $i = \ell - 1$ **downto** 0 **do**
3. $\quad R \leftarrow [2^k]R$
4. $\quad$ **if** $n_i \neq 0$ **then** $R \leftarrow R + [n_i]P$
5. **return** R

Window Methods

Example. Take $n = 2657$ and $k = 3$

We have $n = (5\,1\,4\,1)_8$, and

$$[2657]P = [2^3]([2^3]([2^3][5]P + P) + [4]P) + P$$

Remark.

The 2^k-ary method belongs to the family of window methods

Window Methods

Starting from the binary representation of n,

$$(101001100001)_2$$

simply consider 3bit windows

Window Methods

$(\underset{5}{\boxed{101}}001100001)_2$ $[5]P$

$(101\underset{1}{\boxed{001}}100001)_2$ $[2^3][5]P + P$

$(101001\underset{4}{\boxed{100}}001)_2$ $[2^3]([2^3][5]P + P) + [4]P$

$(101001100\underset{1}{\boxed{001}})_2$ $[2657]P$

$$[2^3]([2^3]([2^3][5]P + P) + [4]P) + P$$

Window Methods

Remark.

If we precompute and store $[n_i]P$, we can speed-up the whole process provided k is suitably chosen

Window Methods

Exercises.

Show that it is sufficient to precompute the odd multiples of P: $[3]P, \ldots, [2^k - 1]P$

Given k, what is the complexity of the 2^k-ary method?

Is there a similar generalization of the right-to-left binary method?

Window Methods

With the 2^k-ary method, the binary expansion of n is sliced in chunks of k bits

To obtain a slightly faster method, let the window slide!

More precisely, as long as the current bit is 0, the window moves right

When a 1 is detected, consider the longest window finishing with 1 whose length is less than or equal to k

Window Methods

$(\underbrace{\boxed{101}}_{5}001100001)_2$ $\qquad [5]P$

$(10100\underbrace{\boxed{11}}_{3}00001)_2$ $\qquad [2^4][5]P + [3]P$

$(101001100000\underbrace{\boxed{1}}_{1})_2$ $\qquad [2^5]([2^4][5]P + [3]P) + P$

instead of

$$[2^3]([2^3]([2^3][5]P + P) + [4]P) + P$$

Window Methods

Remark. Complexity of the sliding window method

On top of the computation of $[3]P, \ldots, [2^k - 1]P$ which can be done with 1 doubling and $2^{k-1} - 1$ additions, we need

- $\log_2 n$ doublings
- $\frac{1}{k+1} \log_2 n$ additions on average

Signed-Digit Representations

To further decrease the number of additions when computing $[n]P$, reduce the density of the expansion of n

We can use signed-coefficients for that

For instance,

$$31 = (11111)_2$$

but we have also $31 = 32 - 1$ and so

$$31 = (10000\bar{1})_{\mathrm{SD}}$$

Signed-Digit Representations

A signed-binary representation of n is any expansion such that

$$n = \sum_{i=0}^{\ell-1} n_i 2^i, \quad \text{with} \quad n_i \in \{-1, 0, 1\}$$

Remarks.

It is straightforward to adapt the double-and-add method to deal with signed-binary representations

For each n, there are many different signed-binary representations

Signed-Digit Representations

One class called the Non-Adjacent Form plays a special role

Every integer n has a unique NAF expansion such that $n_i n_{i+1} = 0$

The density of the NAF is $\frac{1}{3}$

Among all the signed-binary representations this number is minimal for the NAF

This representation can be generalized

Signed-Digit Representations

We can also mix signed-digits and window method

The window-NAF method of length w denoted by NAF_w is such that

- n_i is odd or zero
- $|n_i| < 2^{w-1}$
- there is at most 1 nonzero digit among any w adjacent coefficients

Signed-Digit Representations

Algorithm. NAF_w representation

INPUT: A positive integer n and a parameter $w > 1$.
OUTPUT: The NAF_w representation of n.

1. $i \leftarrow 0$
2. **while** $n > 0$ **do**
3. **if** n is odd **then**
4. $n_i \leftarrow n \bmod 2^w$
5. $n \leftarrow n - n_i$
6. **else** $n_i \leftarrow 0$
7. $n \leftarrow n/2$ and $i \leftarrow i + 1$
8. **return** $(n_{\ell-1} \ldots n_0)_{\mathrm{NAF}_w}$

Signed-Digit Representations

Remarks.

n mods 2^w stands for the smallest residue of n modulo 2^w in absolute value

We need $2^{w-2} - 1$ precomputations

The classical NAF corresponds to $w = 2$

The density of the NAF_w is $\frac{1}{w+1}$

Signed-Digit Representations

Example. $n = 841232$ ($\overline{d}$ denotes $-d$)

$$(n)_2 = 0\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0$$

$$\mathrm{NAF}_2(n) = 1\,0\,\overline{1}\,0\,1\,0\,0\,\overline{1}\,0\,\overline{1}\,0\,\overline{1}\,0\,0\,0\,0\,1\,0\,0\,0\,0$$

$$\mathrm{NAF}_3(n) = 1\,0\,0\,0\,\overline{3}\,0\,0\,0\,\overline{3}\,0\,0\,\mathbf{3}\,0\,0\,0\,0\,1\,0\,0\,0\,0$$

$$\mathrm{NAF}_4(n) = 0\,0\,\mathbf{3}\,0\,0\,0\,0\,7\,0\,0\,0\,\overline{5}\,0\,0\,0\,0\,1\,0\,0\,0\,0$$

Signed-Digit Representations

Exercise.

Show that the density of the NAF_w is $\frac{1}{w+1}$

Hint: Evaluate the average number of bits eliminated at each step in the algorithm computing the NAF_w. For that, compute the probability that, for a given k Line 7 is executed k times in a row

Signed-Digit Representations

Comparison

Both the sliding window of length w and the NAF_w methods have a density of $1/(w+1)$

The sliding window method requires $2^{w-1}-1$ precomputations, whereas the NAF_w only needs $2^{w-2}-1$

Multi-Scalar Multiplication

For certain protocols, e.g. a signature verification, it is necessary to compute a double-scalar multiplication of the form

$$[n]P + [m]Q$$

Instead of computing $[n]P$ and $[m]Q$ separately, we can try to compute them *simultaneously*

Multi-Scalar Multiplication

One idea, known as Shamir's trick, consists in representing n and m jointly

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} n_{\ell-1} \dots n_0 \\ m_{\ell-1} \dots m_0 \end{pmatrix}$$

Then mimick the double-and-add method...

Multi-Scalar Multiplication

Algorithm. Double-Scalar Multiplication

INPUT: $P, Q \in E$ and $\binom{n}{m} = \binom{n_{\ell-1} \ldots n_0}{m_{\ell-1} \ldots m_0}$.
OUTPUT: $[n]P + [m]Q$

1. $R \leftarrow P_\infty$
2. **for** $i = \ell - 1$ **downto** 0 **do**
3. $\quad R \leftarrow [2]R$
4. $\quad$ **if** $\binom{n_i}{m_i} \neq \binom{0}{0}$ **then** $R \leftarrow R + [n_i]P + [m_i]Q$
5. **return** R

Multi-Scalar Multiplication

Remarks.

It makes sense to precompute $P + Q$ to perform at most one addition at each step

If we use a joint-binary representation the joint density is $\frac{3}{4}$ on average

With signed-digit representations, it is necessary to precompute $P - Q$ as well but this can greatly reduce the joint density

Multi-Scalar Multiplication

For instance the Joint Sparse Form is such that:

- of any three consecutive positions
  at least one is a zero column
- adjacent terms do not have opposite signs
- if $n_{i+1}n_i \neq 0$ then $m_{i+1} = \pm 1$ and $m_i = 0$
  The same holds when we exchange n eand m

The JSF has a joint density of $\frac{1}{2}$

Double-Base Number System

To further speed-up a scalar multiplication, another idea is to use more elementary operations

For instance, optimized triplings

With this choice, it is quite natural to represent the integer n in base 3

It can also be interesting to mix doublings and triplings using the Double-Base Number System

Double-Base Number System

Definition. With the Double Base Number System we represent an integer n as

$$n = \sum_{i=1}^{\ell} \pm 2^{a_i} 3^{b_i}$$

Example. $841232 = 2^7 3^8 + 2^1 3^6 - 2^2 3^2 + 2^1$

In general, DBNS expansions are very sparse

Double-Base Number System

Such a representation always exists and is not unique

There is a greedy algorithm to find a DBNS expansion

The length of the expansion of n returned by this greedy approach is on average

$$O\left(\frac{\log n}{\log \log n}\right)$$

Double-Base Number System

Principle of the greedy algorithm

- Start from $t = n$
- Find at each step the best approximation of t as $z = 2^a 3^b$
- Do $t \leftarrow |t - z|$
- Repeat this operation until t is 0

Double-Base Number System

Example.

$$841232 \quad = \quad 2^7 3^8 + 1424$$

Double-Base Number System

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \end{aligned}$$

Double-Base Number System

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \\ 34 &= 2^2 3^2 - 2 \end{aligned}$$

Double-Base Number System

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \\ 34 &= 2^2 3^2 - 2 \end{aligned}$$

Finally,

$$841232 = 2^7 3^8 + 2^1 3^6 - 2^2 3^2 + 2$$

Double-Base Number System

Let us try to compute $[841232]P$ with this representation

$$841232 = 2^7 3^8 + 2^1 3^6 - 2^2 3^2 + 2$$

Just like for binary representations, we can proceed from right-to-left or left-to-right

Double-Base Number System

$$841232 = 2^7 3^8 + 2^1 3^6 - 2^2 3^2 + 2$$

From right-to-left, we start with $[2]P$, then apply 1 doubling and 2 triplings to get $[2^2 3^2]P$

To get the next term, we cannot use $[2^2 3^2]P$, as we would be obliged to divide by 2

So, if we didn't store $[2^1 3^2]P$ we have to compute $[2^1 3^6]P$ from scratch

It doesn't work!

Double-Base Number System

$$841232 = 2^7 3^8 + 2^1 3^6 - 2^2 3^2 + 2$$

From left-to-right, we start with $[2^6 3^2]P$

Then we add P, and we should apply $2^{-1}3^4$ to the result

It doesn't work either!

Double-Base Number System

That is why DBNS expansions are not used under this form

Indeed, if the two sequences of exponents are not simultaneously decreasing, it seems impossible to use only $\max a_i$ doublings **and** $\max b_i$ triplings

$$n = \sum_{i=1}^{\ell} \pm 2^{a_i} 3^{b_i}$$

Double-Base Number System

That is why DBNS expansions are not used under this form

Indeed, if the two sequences of exponents are not simultaneously decreasing, it seems impossible to use only $\max a_i$ doublings **and** $\max b_i$ triplings

That is why the concept of double-base chain has been introduced where one asks also:

$$a_1 \geqslant a_2 \geqslant \cdots \geqslant a_\ell \quad \text{and} \quad b_1 \geqslant b_2 \geqslant \cdots \geqslant b_\ell$$

Double-Base Number System

The greedy algorithm can easily be modified to compute such a DB-chain

Double-Base Number System

The greedy algorithm can easily be modified to compute such a DB-chain

Example.

$$841232 = 2^7 3^8 + 1424$$

Double-Base Number System

The greedy algorithm can easily be modified to compute such a DB-chain

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \end{aligned}$$

Double-Base Number System

The greedy algorithm can easily be modified to compute such a DB-chain

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \\ 34 &= 2^2 3^2 - 2 \end{aligned}$$

Double-Base Number System

The greedy algorithm can easily be modified to compute such a DB-chain

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \\ 34 &= 3^3 + 7 \end{aligned}$$

Double-Base Number System

The greedy algorithm can easily be modified to compute such a DB-chain

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \\ 34 &= 3^3 + 7 \\ 7 &= 3^2 - 2 \end{aligned}$$

Double-Base Number System

The greedy algorithm can easily be modified to compute such a DB-chain

Example.

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424 \\ 1424 &= 2^1 3^6 - 34 \\ 34 &= 3^3 + 7 \\ 7 &= 3^2 - 2 \\ 2 &= 3^1 - 1 \end{aligned}$$

Double-Base Number System

So

$$841232 = 2^7 3^8 + 2^1 3^6 - 3^3 - 3^2 + 3^1 - 1$$

We deduce that

$$[841232]P \; =$$

$$[3]\big([3]\big([3]\big([2^1 3^3]([2^6 3^2]P + P) - P\big) - P\big) + P\big) - P$$

The right-to-left approach works as well

Double-Base Number System

Remark. It is not known if the average length of double-base chains returned by this modified greedy algorithm is still

$$O\left(\frac{\log n}{\log \log n}\right)$$

But there are some heuristics against that

Montgomery's method

Let $P = (x_1, y_1)$ be a point on

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

defined over $\mathbb{F}_p$

It is easy to show that the x-coordinate of $[2]P$ only depends on the x-coordinate of P

Montgomery's method

Let $Q = (x_2, y_2)$ on E such that $P \neq \pm Q$, then the x-coordinate of $P + Q$ is

$$\lambda^2 - x_1 - x_2 \text{ with } \lambda = \frac{y_1 - y_2}{x_1 - x_2}.$$

Exercise.

Show that this coordinate only depends on the x-coordinates of P, Q, and $P - Q$

Montgomery's method

Montgomery showed how to get the x-coordinate of $[n]P$ based on this observation

We can double easily

To add 2 points we need to know the x-ccordinate of the difference

What if this difference is constant and equal to P?

Montgomery's method

To obtain $[n]P$, use Montgomery's ladder

Start from (P_∞, P) and suppose we have $\big([m]P, [m+1]P\big)$ at some point

If the current bit of n is 0 compute $\big([2m]P, [2m+1]P\big)$

Otherwise compute $\big([2m+1]P, [2m+2]P\big)$

Montgomery's method

Algorithm. Montgomery's ladder

INPUT: A point P on E and $n = (n_{\ell-1} \ldots n_0)_2$.
OUTPUT: The point $[n]P$.

1. $P_1 \leftarrow P_\infty$ and $P_2 \leftarrow P$
2. **for** $i = \ell - 1$ **downto** 0 **do**
3. **if** $n_i = 0$ **then**
4. $P_2 \leftarrow P_1 + P_2$ and $P_1 \leftarrow [2]P_1$
5. **else**
6. $P_1 \leftarrow P_1 + P_2$ and $P_2 \leftarrow [2]P_2$
7. **return** P_1

Montgomery's method

Example. Let us compute $[35]P$

$35 = (100011)_2$

Start from (P_∞, P)

The next terms are $(P, [2]P)$ $([2]P, [3]P)$

$([4]P, [5]P)$ $([8]P, [9]P)$ $([17]P, [18]P)$

And finally $([35]P, [36]P)$

Montgomery's method

Complexity. At each step one doubling and one addition is performed

$\log_2 n$ doublings

$\log_2 n$ additions

Montgomery's method

Remarks.

The computation is particularly efficient for a special family of curves

It is possible to recover the y coordinate of $[n]P$, if needed, at the end of the process

This method is naturally immune against side-channel attacks

Koblitz curves

A Koblitz curve E_a is an elliptic curve of the form

$$E_a : y^2 + xy = x^3 + ax^2 + 1, \text{ with } a = 0 \text{ or } 1$$

We consider the points on E_a with coordinates in some suitable extension $\mathbb{F}_{2^d}$

The Frobenius map $\phi(x, y) = (x^2, y^2)$ is an endomorphism of $E_a(\mathbb{F}_{2^d})$

Koblitz curves

It is well-known that the Frobenius satisfies the equation

$$\phi^2 + (-1)^a \phi + [2] = [0]$$

Using this equation, scalar multiplications by powers of 2 can be expressed using ϕ

Koblitz curves

For instance, when $a = 1$, one has

$$[2]P = \phi(P) - \phi^2(P)$$

$$[4]P = -\phi^2(P) - \phi^3(P)$$

$$[8]P = -\phi^3(P) + \phi^5(P)$$

$$[16]P = \phi^4(P) - \phi^8(P)$$

Koblitz curves

Let $\tau \in \mathbb{C}$ such that

$$\tau^2 + (-1)^a \tau + 2 = 0$$

It is easy to show that $\mathbb{Z}[\tau]$ is a Euclidean ring

As a result, any element of $\mathbb{Z}[\tau]$ can be written as a polynomial in τ with coefficients 0 or 1

Koblitz curves

Similarly, every integer n has a τ-NAF expansion

$$n = \sum_i n_i \tau^i, \quad \text{with} \quad n_i \in \{0, \pm 1\}$$

and such that $n_i n_{i+1} = 0$

The density of the τ-NAF is $\frac{1}{3}$

Koblitz curves

This implies that

$$[n]P = \sum_i n_i \phi^i(P)$$

and $[n]P$ can be obtained with a Frobenius-and-add method

Scalar multiplications on Koblitz curves are therefore very efficient