Elliptic Scalar Multiplication Using Point Halving

Erik Woodward Knudsen
Oberthur Card Systems
e.knudsen@oberthurcs.com

Elliptic Curve Protocols

The central operation : Scalar multiplication

Standard method : Double-and-add algorithm

Idea : Use the doubling formulas “backwards”

Consequence : Can use “halve-and-add” algorithm

Non-supersingular Elliptic Curves Defined over $\mathbf{F}_{2^n}$

$$E = \{(x,y) \in \overline{F}_{2^n} \times \overline{F}_{2^n} \mid y^2+xy = x^3+ax^2+b\} \cup \{\mathcal{O}\};\ a,b \in \mathbf{F}_{2^n},\ b \neq 0$$

$$E(\mathbf{F}_{2^n}) = \{(x,y) \in \mathbf{F}_{2^n} \times \mathbf{F}_{2^n} \mid y^2 + xy = x^3 + ax^2 + b\} \cup \{\mathcal{O}\}$$

Equipped with an abelian group structure

Non-supersingular Elliptic Curves Defined over $\mathbf{F}_{2^n}$

m-torsion points : $E[m] = \{P \in E | [m]P = \mathcal{O}\}$

Structure : $\forall k \in \mathbf{N}$: $E[2^k] \simeq \mathbf{Z}/2^k\mathbf{Z}$

We have in fact :

$$E(\mathbf{F}_{2^n}) = G \times E[2^k]$$

where $k \geq 1$ and $|G|$ is odd

$[2] : G \to \; G$ bijection on $G \subset E(\mathbf{F}_{2^n})$

The aim : Compute its inverse.

The halving map $[\frac{1}{2}] : G \to G$

Doubling Formulas

$$\begin{aligned}
\lambda_Q &= u + \frac{v}{u} \\
x &= \lambda_Q^2 + \lambda_Q + a \\
y &= (x + u)\lambda_Q + x + v
\end{aligned}$$

Input : $Q = (u, v)$

Output : $P = (x, y) \quad$ where $[2]Q = P$

Halving Formulas

$$\begin{aligned}
\lambda_Q^2 + \lambda_Q &= a + x && (i)\\
u^2 &= x(\lambda_Q + 1) + y = x(\lambda_Q + \lambda_P + x + 1) && (ii)\\
v &= u(u + \lambda_Q) && (iii)
\end{aligned}$$

Input : $P = (x, y) \in G$
Output : $Q = (u, v)$
where

$$Q = [\tfrac{1}{2}]P \in G$$

or

$$Q = [\tfrac{1}{2}]P + T_2 \in E(\mathbf{F}_{2^n}) \backslash G$$

The Curve $E(\mathbf{F}_{2^n}) = G \times E[2]$

$$P \in G,\ [2]Q = P,\ [2]Q_1 = Q = (u, v)$$

Necessary and sufficient conditions :

$$Q = [\tfrac{1}{2}]P$$

$$\Updownarrow$$

$$Q_1 \in E(\mathbf{F}_{2^n})$$

$$\Updownarrow$$

$$\exists \lambda \in \mathbf{F}_{2^n} : \lambda^2 + \lambda = a + u$$

Point Halving Algorithm for $E(\mathbf{F}_{2^n}) = G \times E[2]$

Input : $P \in G$ represented either as (x, y) or as (x, λ_P)
Output : $[\frac{1}{2}]P \in G$ represented as $(u, \lambda_{[\frac{1}{2}]P})$
Method :
1. Compute a $\lambda_{[\frac{1}{2}]P}$ such that $\lambda^2_{[\frac{1}{2}]P} + \lambda_{[\frac{1}{2}]P} = a + x$
2. Compute $u^2 = x(\lambda_{[\frac{1}{2}]P} + 1) + y = x(\lambda_{[\frac{1}{2}]P} + \lambda_P + x + 1)$
3. Check if there exists a $\lambda \in \mathbf{F}_{2^n}$ such that $\lambda^2 + \lambda = a + u$.
4. If such a λ does not exist then compute
$u^2 := u^2 + x$ and $\lambda_{[\frac{1}{2}]P} := \lambda_{[\frac{1}{2}]P} + 1$
5. Calculate $u := \sqrt{u^2}$
6. Output $(u, \lambda_{[\frac{1}{2}]P})$

The Curve $E(\mathbf{F}_{2^n}) = G \times E[2]$

The algorithm requires

- solving 1 second degree equation (in 1)
- 1 multiplication (in 2)
- 1 check (in 3)
- 1 square root (in 5)

and if v is to be evaluated using (iii) one extra multiplication is required.

Applications for Scalar Multiplication

$$c \in \mathbf{Z},\ \ P \in G,\ \ ord(P) = r,\ \ m = \lfloor log_2(r) \rfloor$$

$$2^m c \pmod r = \sum_{i=0}^{m} \widehat{c}_i 2^i \qquad \widehat{c}_i \in \{0,1\}$$

$$\sum_{i=0}^{m} \frac{c_i}{2^i} := \sum_{i=0}^{m} \widehat{c}_i 2^{i-m} \qquad c_i \in \{0,1\}$$

An Arbitrary Curve $E(\mathbf{F}_{2^n}) = G \times E[2^k]$

$$P \in G,\ [2]Q = P,\ [2]Q_1 = Q$$

$$[2]Q_i = Q_{i-1},\ 2 \le i \le k$$

Necessary and sufficient condition :

$$Q = [\frac{1}{2}]P \Leftrightarrow Q_k \in E(\mathbf{F}_{2^n})$$

An Arbitrary Curve $E(\mathbf{F}_{2^n}) = G \times E[2^k]$

The algorithm requires :

- solving k second degree equations
- k multiplications
- 1 check
- k or $k+1$ square roots

Computing Efficiently

- the check

- the square root

- solving a second degree equation

Normal Basis

Easy !

Polynomial Basis

The check :

$$\exists \lambda \in \mathbf{F}_{2^n} : \lambda^2 + \lambda = a + u$$

$$\Updownarrow$$

$$Tr(a) = Tr(u^2)$$

Storage : 1 field element

Time : negligible

Polynomial Basis

$$x = \sum_{i=0}^{n-1} x_i T^i = \sum_{i\ even} x_i T^i + T \sum_{i\ odd} x_i T^{i-1} \qquad x_i \in \{0,1\}$$

Square root :

$$\sqrt{x} = \sum_{i\ even} x_i T^{\frac{i}{2}} + \sqrt{T} \sum_{i\ odd} x_i T^{\frac{i-1}{2}}$$

Storage : 1 field element.
Time : app. $\frac{1}{2}$ field multiplication

Polynomial Basis

Solving a Second Degree Equation : $\frac{1}{2}$ matrix multiplication

Storage : $\frac{n^2}{2}$ bits $\sim \frac{n}{2}$ field elements

Time : $\frac{n}{4}$ field additions $\sim \frac{1}{2}$ field multiplication

Expected Performance

Length of bitvector : m
Number of non-zero entries : l

operation	double-and-add	halve-and-add
inversions	$m+l$	l
multiplications	$2m+2l$	$m+3l$
squarings	$m+l$	l
solving $\lambda^2+\lambda=a+x$	0	m
square roots	0	m
checks	0	m

Normal Basis : 67% improvement
Polynomial Basis : 39% improvement