Elliptic-curve cryptography IX

Explicit formulas

Tanja Lange

Eindhoven University of Technology

2MMC10 – Cryptology

How to implement curve arithmetic in practice?

In $\mathbf{F}_p$ with large p divisions are a lot more expensive than multiplications.

How to implement curve arithmetic in practice?

In $\mathbf{F}_p$ with large p divisions are a lot more expensive than multiplications.

We use projective coordinates to delay inversions:
Use $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ to represent $(x_1, y_1) = (X_1/Z_1, Y_1/Z_1)$,
i. e., $(X_1 : Y_1 : Z_1) = (\lambda X_1 : \lambda Y_1 : \lambda Z_1)$ for $\lambda \neq 0$.
Delay division till the end of scalar multiplication aP.

How to implement curve arithmetic in practice?

In $\mathbf{F}_p$ with large p divisions are a lot more expensive than multiplications.

We use projective coordinates to delay inversions:
Use $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ to represent $(x_1, y_1) = (X_1/Z_1, Y_1/Z_1)$,
i. e., $(X_1 : Y_1 : Z_1) = (\lambda X_1 : \lambda Y_1 : \lambda Z_1)$ for $\lambda \neq 0$.
Delay division till the end of scalar multiplication aP.

Derive formulas stating with $(x_i, y_i) = (X_i/Z_i, Y_i/Z_i)$,
bring result onto same denominator Z_3.

How to implement curve arithmetic in practice?

In $\mathbf{F}_p$ with large p divisions are a lot more expensive than multiplications.

We use projective coordinates to delay inversions:
Use $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ to represent $(x_1, y_1) = (X_1/Z_1, Y_1/Z_1)$,
i.e., $(X_1 : Y_1 : Z_1) = (\lambda X_1 : \lambda Y_1 : \lambda Z_1)$ for $\lambda \neq 0$.
Delay division till the end of scalar multiplication aP.

Derive formulas stating with $(x_i, y_i) = (X_i/Z_i, Y_i/Z_i)$,
bring result onto same denominator Z_3.

Feature: These formulas capture ∞ on Weierstrass curve as $(0 : 1 : 0)$.

How to implement curve arithmetic in practice?

In $\mathbf{F}_p$ with large p divisions are a lot more expensive than multiplications.

We use projective coordinates to delay inversions:
Use $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ to represent $(x_1, y_1) = (X_1/Z_1, Y_1/Z_1)$,
i. e., $(X_1 : Y_1 : Z_1) = (\lambda X_1 : \lambda Y_1 : \lambda Z_1)$ for $\lambda \neq 0$.
Delay division till the end of scalar multiplication aP.

Derive formulas stating with $(x_i, y_i) = (X_i/Z_i, Y_i/Z_i)$,
bring result onto same denominator Z_3.

Feature: These formulas capture ∞ on Weierstrass curve as $(0 : 1 : 0)$.

Sometimes cheaper to keep separate denominators:
$(x_i, y_i) = (X_i/Z_i, Y_i/T_i)$ represented as $((X : Z), (Y : T))$.

How to implement curve arithmetic in practice?

In $\mathbf{F}_p$ with large p divisions are a lot more expensive than multiplications.

We use projective coordinates to delay inversions:
Use $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ to represent $(x_1, y_1) = (X_1/Z_1, Y_1/Z_1)$,
i. e., $(X_1 : Y_1 : Z_1) = (\lambda X_1 : \lambda Y_1 : \lambda Z_1)$ for $\lambda \neq 0$.
Delay division till the end of scalar multiplication aP.

Derive formulas stating with $(x_i, y_i) = (X_i/Z_i, Y_i/Z_i)$,
bring result onto same denominator Z_3.

Feature: These formulas capture ∞ on Weierstrass curve as $(0 : 1 : 0)$.

Sometimes cheaper to keep separate denominators:
$(x_i, y_i) = (X_i/Z_i, Y_i/T_i)$ represented as $((X : Z), (Y : T))$.

This is also the best way to see points at infinity on Edwards curves

$$((1 : 0), (\pm\sqrt{d} : \sqrt{a})) \text{ and } ((1 : \pm\sqrt{d}), (1 : 0))$$

if these exist.

Projective coordinates for Edwards curves

Taking inputs $P_1 = (X_1 : Y_1 : Z_1), P_2 = (X_2 : Y_2 : Z_2)$,
producing $P_1 + P_2 = P_3 = (X_3 : Y_3 : Z_3)$.

Optimized formulas:

$$\begin{aligned}
A &= Z_1 \cdot Z_2;\ B = A^2;\ C = X_1 \cdot X_2;\ D = Y_1 \cdot Y_2;\\
E &= d \cdot C \cdot D;\ F = B - E;\ G = B + E;\\
X_3 &= A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D);\\
Y_3 &= A \cdot G \cdot (D - C);\\
Z_3 &= F \cdot G.
\end{aligned}$$

Needs 10M + 1S + 1d-mult + 7add.

Projective coordinates for Edwards curves

Taking inputs $P_1 = (X_1 : Y_1 : Z_1), P_2 = (X_2 : Y_2 : Z_2)$, producing $P_1 + P_2 = P_3 = (X_3 : Y_3 : Z_3)$.

Optimized formulas:

$$\begin{aligned}
A &= Z_1 \cdot Z_2;\ B = A^2;\ C = X_1 \cdot X_2;\ D = Y_1 \cdot Y_2; \\
E &= d \cdot C \cdot D;\ F = B - E;\ G = B + E; \\
X_3 &= A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D); \\
Y_3 &= A \cdot G \cdot (D - C); \\
Z_3 &= F \cdot G.
\end{aligned}$$

Needs $10\mathbf{M} + 1\mathbf{S} + 1\mathbf{d}\text{-mult} + 7\mathbf{add}$.

See the EFD for many more formulas and the whole zoo of curve shapes.

As designer choose curves with small constants (under the condition that the system is secure – we will see what that means soon).

Reminder: Montgomery ladder

```
def cswap(bit, R, S):  # constant time conditional swap
        dummy = bit * (R - S)   # 0 or R - S
        R = R - dummy    # R or R - (R - S) = S
        S = S + dummy    # S or S + (R - S) = R
        return (R, S)

a = 44444   # our super secret scalar. No, not that one.
l = max    # some maximum bit length, matching order(P)
A = a.digits(2,padto = l) # fill with 0 to lenght l
P0 = 0     # so initial doublings don't matter, 0=0P
P1 = P    # difference P1 - P0 = P
for i in range(l-1,-1,-1): # fixed-length loop
  (P0, P1) = cswap(A[i], P0, P1)  # see above
  P1 = P0 + P1  # addition with fixed difference
  P0 = 2P0  # double point for which bit is set
  (P0, P1) = cswap(A[i], P0, P1) # swap back, can merge
print(P0)
```

This uses one doubling and one differential addition per bit.

Montgomery differential addition

Let $nP = (U_n : V_n : Z_n), mP = (U_m : V_m : Z_m)$ with known difference $(m - n)P = (U_{m-n} : V_{m-n} : Z_{m-n})$ on

$$M_{A,B} : Bv^2 = u^3 + Au^2 + u.$$

We will only use U and Z; cheaper by skipping V.

Montgomery differential addition

Let $nP = (U_n : V_n : Z_n), mP = (U_m : V_m : Z_m)$ with known difference $(m-n)P = (U_{m-n} : V_{m-n} : Z_{m-n})$ on

$$M_{A,B} : Bv^2 = u^3 + Au^2 + u.$$

We will only use U and Z; cheaper by skipping V.

Addition: $n \neq m$

$$\begin{aligned} U_{m+n} &= Z_{m-n}\big((U_m - Z_m)(U_n + Z_n) + (U_m + Z_m)(U_n - Z_n)\big)^2, \\ Z_{m+n} &= U_{m-n}\big((U_m - Z_m)(U_n + Z_n) - (U_m + Z_m)(U_n - Z_n)\big)^2 \end{aligned}$$

Doubling: $n = m$

$$\begin{aligned} 4U_nZ_n &= (U_n + Z_n)^2 - (U_n - Z_n)^2, \\ U_{2n} &= (U_n + Z_n)^2(U_n - Z_n)^2, \\ Z_{2n} &= 4U_nZ_n\big((U_n - Z_n)^2 + \big((A+2)/4\big)(4U_nZ_n)\big). \end{aligned}$$

Differential addition takes 4M and 2S. Doubling takes 3M and 2S.

Montgomery differential addition

Let $nP = (U_n : V_n : Z_n), mP = (U_m : V_m : Z_m)$ with known difference $(m-n)P = (U_{m-n} : V_{m-n} : Z_{m-n})$ on

$$M_{A,B} : Bv^2 = u^3 + Au^2 + u.$$

We will only use U and Z; cheaper by skipping V.

Addition: $n \neq m$

$$\begin{aligned} U_{m+n} &= Z_{m-n}\big((U_m - Z_m)(U_n + Z_n) + (U_m + Z_m)(U_n - Z_n)\big)^2, \\ Z_{m+n} &= U_{m-n}\big((U_m - Z_m)(U_n + Z_n) - (U_m + Z_m)(U_n - Z_n)\big)^2 \end{aligned}$$

Doubling: $n = m$

$$\begin{aligned} 4U_nZ_n &= (U_n + Z_n)^2 - (U_n - Z_n)^2, \\ U_{2n} &= (U_n + Z_n)^2(U_n - Z_n)^2, \\ Z_{2n} &= 4U_nZ_n\big((U_n - Z_n)^2 + \big((A+2)/4\big)(4U_nZ_n)\big). \end{aligned}$$

Differential addition takes 4M and 2S. Doubling takes 3M and 2S.
In ladder, $m - n = 1$, choose $Z_{m-n} = 1$ and $(A+2)/4$ small.
Then cost per bit: 5M and 4S. Also like U_{m-n} small.

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from
`http://safecurves.cr.yp.to/`.

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from http://safecurves.cr.yp.to/.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from http://safecurves.cr.yp.to/.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Note that the map given in part VI maps to $a'x'^2 + y'^2 = 1 + d'x'^2y'^2$ with $a' = 486664, d' = 486660.$

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1$.

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from
`http://safecurves.cr.yp.to/`.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Note that the map given in part VI maps to $a'x'^2 + y'^2 = 1 + d'x'^2y'^2$
with $a' = 486664, d' = 486660$.
Note $a' = b^2$ in $\mathbf{F}_p$ and change $x = bx'$, $y = y'$.

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from
http://safecurves.cr.yp.to/.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Note that the map given in part VI maps to $a'x'^2 + y'^2 = 1 + d'x'^2y'^2$
with $a' = 486664, d' = 486660.$
Note $a' = b^2$ in $\mathbf{F}_p$ and change $x = bx'$, $y = y'$.
This maps to $x^2 + y^2 = 1 + dx^2y^2$ with $d = d'/a' = 121665/121666.$