Faster point scalar multiplication on NIST elliptic curves over GF(p) using (twisted) Edwards curves over GF(p^3)

Michał WROŃSKI

Institute of Mathematics and Cryptology
Department of Cybernetics
Military University of Technology in Warsaw, Poland
mwronski@wat.edu.pl

Abstract

In this paper we present a new method for fast scalar multiplication on elliptic curves over $GF(p)$ in FPGA using Edwards and twisted Edwards curves over $GF(p^3)$. The presented solution works for curves with prime group order (for example for all NIST curves over $GF(p)$). It is possible because of using 2-isogenous twisted Edwards curves over $GF(p^3)$ instead of using short Weierstrass curves over $GF(p)$ for point scalar multiplication. This problem was considered by Verneuil in [1], but in software solutions it is useless, because multiplication in $GF(p^3)$ is much harder than multiplication in $GF(p)$. Fortunately in hardware solutions it is possible to make in FPGA fast multiplication in $GF(p^3)$ using parallel computations. Single multiplication in $GF(p^3)$ is still a little bit slower than in $GF(p)$ but operations on twisted Edwards curves require less multiplications than operations on short Weierstrass curves. Using these observations results in that scalar multiplication on twisted Edwards curve may be in some situations shorter than scalar multiplication on short Weierstrass curve up to 26%. Moreover, in Edwards and twisted Edwards curves arithmetic it is possible to use unified formula (the same formula for points addition and point doubling) which protects us against some kinds of side channel attacks. We also present full coprocessor for fast scalar multiplication in FPGA using described techniques.

Keywords. Edwards curves. Twisted Edwards curves. Finite fields. Point scalar multiplication.

1 Introduction

We present a new method for fast scalar multiplication on elliptic curves over $GF(p)$ in FPGA using Edwards and twisted Edwards curves over $GF(p^3)$. The presented solution works for all elliptic curves given by short Weierstrass equation, especially for all NIST curves over $GF(p)$. In Edwards and twisted Edwards curves arithmetic it is possible to use unified formula. It is well known

that unified formulas (the same formula for point doubling and points addition) exist for some special types of elliptic curves, for example Edwards and twisted Edwards curves. Unfortunately, till now it was impossible to use such unified formula for NIST curves over $GF(p)$, because they are not NIST curves isomorphic to any special kind of elliptic curve for which unified formula exists.

Using Edwards and twisted Edwards curves which are 2-isogenous to short Weierstrass curves was considered in [1], because it is easy to make transformations between points on curves which are 2-isogenous. Unfortunately, twisted Edwards curve which is 2-isogenous to short Weierstrass curve over $GF(p)$ exists if such short Weierstrass curve has three points of order two. This condition for most curves does not occur. To avoid this problem, we need to use field extension from $GF(p)$ to $GF(p^3)$.

2 Elliptic curve arithmetic

However point scalar multiplication on elliptic curve requires a lot of computations, it is not computational hard. Often point on elliptic curve is given in affine coordinates. Unfortunately, counting scalar multiplication in affine coordinates requires counting inversion of element which is the most costly operation. That is why the other coordinates systems are searched to be used instead of affine coordinates. In this case the best is when scalar multiplication is inversion free and requires small number of multiplications, because multiplication is much more costly than addition/subtraction. Because of reason described above the most popular are projective coordinates which allow us to make scalar multiplication without inversions (instead of the one inversion we often make to transform point from projective coordinates to affine coordinates).
There are some special kinds of elliptic curves on which it is possible to make scalar multiplication much faster than on short Weierstrass curve but they are useless if we want to use curve which has prime order of points group. The reason of this fact is that order of points group on Edwards curves, Montgomery curves or twisted Edwards curves and another which allow faster arithmetic is never prime number. In other words, it is impossible for every short Weierstrass curve to find isomorphic Edwards curves, Montgomery curves or twisted Edwards curves but it is always possible to do it in opposite way.
There is another disadvantage of arithmetic on Weierstrass curve. The points addition and point doubling must be done using different formulas what makes point scalar multiplication vulnerable for side channel attacks. This danger always exists if we use point scalar multiplication methods similar to double-and-add method, when for bit equal 0 we make only doubling and for bit equal 1 we make doubling and after that adding. Of course using Montgomery ladder let us to avoid this problem. Unfortunately if we use this kind of point scalar multiplication then for every bit of number k for which we compute $Q = [k]P$ we need to do one point addition and one point doubling. That is why Montgomery ladder is more costly.
Edwards curves and twisted Edwards curves allow us to use unified formula

which means that we may use the same formula for points addition and point doubling. So the question is if it is some possibility to use arithmetic on Edwards or twisted Edwards curves for every short Weierstrass curve, especially for NIST curves?
Verneuil in [1] showed that we may use arithmetic on Edwards or twisted Edwards curves for every short Weierstrass curve if we use the field extension from $GF(p)$ to $GF(p^3)$ for twisted Edwards curves and at least to $GF(p^6)$ for Edwards curves then It is possible using 2-isogenous twisted Edwards or Edwards curves to short Weierstrass curves. We will show that for all NIST curves instead of NIST P-224 it is possible to use Edwards curve arithmetic using field extension from $GF(p)$ to $GF(p^3)$.

It is also obvious that arithmetic in $GF(p^3)$ or in $GF(p^6)$ is much more costly than in $GF(p)$. That is why using this arithmetic is useless in software solutions even if the number of operations in points addition or point doubling on twisted Edwards curve and Edwards curve is much smaller.
The things look much different if we consider the hardware solutions. In this case we are able to make parallelization of most of operations which results in that multiplication in $GF(p^3)$ is not much longer than in $GF(p)$.

We present the first FPGA coprocessor with unified formula for NIST curves over $GF(p)$.

2.1 Scalar multiplication on short Weierstrass curve

Lange and Bernstein in [2] present the best formulas for point addition and point doubling on short Weierstrass curves over $GF(p)$.

Short Weierstrass curve for every field with characteristic coprime with 6 is given by equation:
$E : y^2 = x^3 + ax + b$ in affine coordinates
or $E : Y^2Z = X^3 + aXZ^2 + bZ^3$ in projective coordinates.

Because if we use affine coordinates we need to compute inversion of element after every single point addition or point doubling we use projective coordinates (or similar) which are inversion-free (there is required one inversion at the end of all computations) and they are more efficient.
Points addition of $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ in projective coordinates may be computed using formulas:

1. $Y1Z2 = Y_1 \cdot Z_2$
2. $X1Z2 = X_1 \cdot Z_2$
3. $Z1Z2 = Z_1 \cdot Z_2$
4. $u = Y_2 \cdot Z_2 - Y1Z2$
5. $uu = u^2$
6. $v = X_2 \cdot Z_1 - X1Z2$

7. $vv = v^2$
8. $vvv = v \cdot vv$
9. $R = vv \cdot X1Z2$
10. $A = uu \cdot Z1Z2 - vvv - 2 \cdot R$
11. $X_3 = v \cdot A$
12. $Y_3 = u \cdot (R - A) - vvv \cdot Y1Z2$
13. $Z_3 = vvv \cdot Z1Z2$

Then $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$. In points addition sometimes it is useful to assume that $Z_2 = 1$. In this case we need not to count $Z1Z2 = Z_1 \cdot Z_2$ and we have one multiplication less.

The doubling of the point $P_1 = (X_1, Y_1, Z_1)$ may be computed using formulas:

1. $XX = X_1^2$
2. $ZZ = Z_1^2$
3. $w = a \cdot ZZ + 3 \cdot XX$
4. $s = 2 \cdot Y_1 \cdot Z_1$
5. $ss = s2$
6. $sss = s \cdot ss$
7. $R = Y1 \cdot s$
8. $RR = R^2$
9. $B = (X_1 + R)^2 - XX - RR$
10. $h = w^2 - 2 \cdot B$
11. $X_3 = h \cdot s$
12. $Y_3 = w \cdot (B - h) - 2 \cdot RR$
13. $Z_3 = sss$

Then $P_3 = (X_3, Y_3, Z_3) = [2]P_1$

It is easy to see that points addition requires 14 multiplications (12 multiplications and 2 squares) and 7 additions/subtractions (or 6 additions and one multiplication by 2).

The point doubling requires 12 multiplications (5 multiplications, 6 squares and 1 multiplication by constant) and 12 additions/subtractions (7 additions/subtractions, 3 multiplications by 2 and 1 multiplication by 3).

3 Edwards curves and twisted Edwards curves

Edwards and twisted Edwards curves are described with many additional details in [3], [4] and [5]. Below we present only the most important information about Edwards and twisted Edwards curves.

3.1 Edwards curves

Edwards curve over field K with characteristic not equal 2 is given by formula:

$E_e : x^2 + y^2 = 1 + dx^2y^2$, where $d \in K\backslash\{0, 1\}$

For every Edwards curve exists birationally equivalent short Weierstrass curve but not for every short Weierstrass curve exists birationally equivalent Edwards curve. The sum of two points in affine coordinates $(x_1, y_1), (x_2, y_2)$ on curve E_e is given by:

$P + Q = (\frac{x_1y_2+y_1x_2}{1+dx_1x_2y_1y_2}, \frac{y_1y_2-x_1x_2}{1-dx_1x_2y_1y_2})$

It is easy to see that point $(0, 1)$ is the neutral element of addition law. Points $(1, 0)$ and $(-1, 0)$ have order 4 and point $(0, -1)$ has order 2. Moreover, the presented addition law is unified: it can be used to double a point and works also for neutral element. If d is nonsquare in K then addition law is complete (works for all pairs of inputs). Using Edwards addition law (especially using inverted coordinates) requires much less multiplications than standard coordinates systems on short Weierstrass curve (like projective coordinates).

3.2 Twisted Edwards curves

$E_t : ax^2 + y^2 = 1 + dx^2y^2$, where $d \in K\backslash\{0, 1\}$

For every twisted Edwards curve exists birationally equivalent short Weierstrass curve but not for every short Weierstrass curve exists birationally equivalent twisted Edwards curve. The sum of two points in affine coordinates $(x_1, y_1), (x_2, y_2)$ on curve E_t is:

$P + Q = (\frac{x_1y_2+y_1x_2}{1+dx_1x_2y_1y_2}, \frac{y_1y_2-ax_1x_2}{1-dx_1x_2y_1y_2})$

In [8] it is shown that however for completeness of these formula a must to be a square in K and d must to be a non-square in K, in some assumptions we may suppose that it is complete, especially if K is a field of odd characteristic and we have a given twisted Edwards curve over K then if points $P = (x_1, y_1), Q = (x_2, y_2)$ on this curve are of odd order, then $1 - dx_1x_2y_1y_2 \neq 0$ and $1 + dx_1x_2y_1y_2 \neq 0$ and $P + Q$ cannot be in this case point at infinity.

It means that we need not to worry about if a or d are squares or non-squares because all NIST curves are defined over field of odd characteristic and order of twisted Edwards curve is always even. So any of the points: $P, Q, P + Q$ cannot be the point at infinity. It means that we can use twisted Edwards curve for scalar multiplication points of prime groups without any exceptions.

3.3 Arithmetic in inverted coordinates

The fastest formulas for points addition and point doubling have been found for Edwards and twisted Edwars curves. The best results may be achieved using in both cases inverted coordinates. They require small amount of multiplication and they are inversion free.

Let's consider the point $P = (X_1, Y_1, Z_1)$ in inverted coordinates on the twisted Edwards curve: $(X^2 + aY^2)Z^2 = X^2Y^2 + dZ^4$ Where $X_1, Y_1, Z_1 \neq 0$ then point in affine coordinates on curve E_t is given by: $(\frac{Z_1}{X_1}, \frac{Z_1}{Y_1})$. We show below the algorithm for point addition

$P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$, $R = P + Q = (X_3, Y_3, Z_3)$.

1. $A = Z_1 \cdot Z_2$
2. $B = d \cdot A^2$
3. $C = X_1 \cdot X_2$
4. $D = Y_1 \cdot Y_2$
5. $E = C \cdot D$
6. $H = C - a \cdot D$
7. $I = (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D$
8. $X_3 = (E + B) \cdot H2$
9. $Y_3 = (E - B) \cdot I$
10. $Z_3 = A \cdot H \cdot I$

We can use presented formulas to make addition of points even if $P = Q$. We can say that this formula is unified and it requires 12 multiplications (8 multiplications, 1 squaring and 2 multiplications by constants) and 7 additions/subtractions. Sometimes we can assume that $Z_1 = 1$. Then algorithm requires 1 multiplication less.

It is possible to speed-up the algorithm using different formulas for point addition and different formulas for point doubling. Unfortunately in this case the device used for these computations is vulnerable for side channel attacks.

The algorithm for point doubling $R = (X_3, Y_3, Z_3) = [2]P$, where $(d_2 = 2 \cdot d)$:

1. $A = X_1^2$
2. $B = Y_1^2$
3. $U = a \cdot B$

4. $C = A + U$
5. $D = A - U$
6. $E = (X_1 + Y_1)^2 - A - B$
7. $X_3 = C \cdot D$
8. $Y_3 = E \cdot (C - d_2 \cdot Z_1^2)$
9. $Z_3 = D \cdot E$

The algorithm requires 9 multiplications (3 multiplications, 4 squares, 2 multiplications by constants) and 6 additions/subtractions.

Moreover, in both algorithms if we put $a = 1$ then we obtain arithmetic on Edwards curve and algorithm for point doubling and algorithm for points addition require one multiplication less.

3.4 Isomorphism between Edwards and twisted Edwards curves

If a or d is square in K, then for twisted Edwards curve we can find birational equivalent Edwards curve.

For given twisted Edwards curve $E_t : ax^2 + y^2 = 1 + dx^2y^2$

I. If d is square in K, then we can make transformations $(x, y) \rightarrow (x', y') = (x\sqrt{d}, 1/y)$
If we make substitution:
$x = \frac{x'}{\sqrt{d}}, y = \frac{1}{y'}$ we get in result:
$\frac{a}{d}(x')^2 + \frac{1}{(y')^2} = 1 + (\frac{x'}{y'})^2$ and then
$\frac{a}{d}(x')^2(y')^2 + 1 = (y')^2 + (x')^2$
$(x')^2 + (y')^2 = 1 + \frac{a}{d}(x')^2(y')^2$

II. If a is square in K, then:
$(x, y) \rightarrow (x', y') = (x\sqrt{a}, y)$
If we make substitution:
$x = \frac{x'}{\sqrt{a}}, y = y'$ we get in result:
$(x')^2 + (y')^2 = 1 + \frac{d}{a}(x')^2(y')^2$

These formulas are so useful, because arithmetic on Edwards curves requires fewer multiplications than arithmetic on twisted Edwards curve. For all NIST curves instead of NIST P-224 we can use arithmetic on Edwards curve. For NIST P-224 we use twisted Edwards curve arithmetic because both a and d are non-squares in K.

4 Isogenies

Isogeny is almost the same what isomorphism is. In the case of elliptic curves two curves over field K are isomorphic if and only if they have the same order of points group and the same torsion subgroups. Isogeny is a little weaker.

Now we present some useful theorems, proofs may be found in [7] and [9].

Theorem 1 *Isogeny is homomorphism:*
$\phi([k_1]P_1 + [k_2]P_2) = [k_1]\phi(P_1) + [k_2]\phi(P_2)$

Because isogeny is homomorphism it always maps point at infinity into the point at infinity: $\phi(O) = \phi([0]P) = [0]\phi(P) = O$.

Theorem 2 *(Tate) Two curves are isogenous if and only if they have the same order of points group. It means that they may have different torsion subgroups.*

We present below some properties of isogenies that are necessary in our work.

Theorem 3 *Number of elements of isogeny kernel is called degree of isogeny and is denoted by* $\deg(\phi)$.

Theorem 4 *For every isogeny ϕ exists the dual isogeny ϕ' such that $\phi \circ \phi' = \phi' \circ \phi = \deg(\phi)$.*

It means that for 2-isogeny $\phi \circ \phi' = 2$ and then $(\phi \circ \phi')(P) = [2]P$. So if we want to count $[k]P$ using isogenous curves, we need to count $(\phi \circ \phi')([\frac{k}{\deg(\phi)}]P) = [k]P$. If $\deg(\phi) = 2$ then we need to count $(\phi \circ \phi')([\frac{k}{2}]P) = [k]P$.

Basing on these theorems we proved the theorem below:

Theorem 5 *Let P_1 be the point on E_1 and P_2 be the point on isogenous curve E_2 then:*
If $P_2 = \phi(P_1)$ and $n_2 = Ord(P_2), n_1 = Ord(P_1)$, then $n_2 | n_1$.

Proof. Because $\phi([n_1]P_1) = [n_1]\phi(P_1) = [n_1]P_2$ but $\phi([n_1]P_1) = \phi(O) = O$, so

$[n_1]P_2 = O$ and then $n_2 = Ord(P_2) \leq Ord(P_1) = n_1$. Now let's suppose that $n_2 \nmid n_1$ and $m = \left\lfloor \frac{n_1}{n_2} \right\rfloor, r = n_1 \bmod n_2$. Then $\phi([n_2]P_1) = [n_2]\phi(P_1) = [n_2]P_2 = O$.

Next:
$\phi([m \cdot n_2]P_1) = [m \cdot n_2]P_2 = O$ and $\phi([n_1]P_1) = \phi([m \cdot n_2 + r]P_1) = \phi([m \cdot n_2]P_1 + [r]P_1) = \phi([m \cdot n_2]P_1) + \phi([r]P_1) = O + \phi([r]P_1) = \phi([r]P_1) = O$. It cannot be true because $r < n_2$ but n_2 is the smallest number for which $[n_2]P_2 = O$. It means that $n_2 | n_1$. ■

Theorem 6 *Let P_1 be the point on E_1 and P_2 be the point on isogenous curve E_2 then:*
If $P_2 = \phi(P_1)$ and $n_2 = Ord(P_2), n_1 = Ord(P_1)$, then $n_2 \geq \deg(\phi) \cdot n_1$

Proof. $\phi'(\phi([n_2]P_1)) = \phi'([n_2]\phi(P_1)) = [n_2]\phi'(\phi(P_1)) = [n_2 \cdot \deg(\phi)]P_1$. But:
$\phi'(\phi([n_2]P_1)) = \phi'([n_2]\phi(P_1)) = \phi'([n_2]P_2) = \phi'(O) = O$. And finally:
$O = [n_2 \cdot \deg(\phi)]P_1$, which means that $n_2 \cdot \deg(\phi) \geq n_1$. ■

If $n_1 \neq \deg(\phi)$ and n_1 is prime then $n_2 = n_1$ and if we want to use isogenous curve E_2 to count $[k]P_1$ then we must count such k' that $\phi'(\phi([k']P_1)) = \phi'([k']\phi(P_1)) = [k']\phi'(\phi(P_1)) = [k' \cdot \deg(\phi)]P_1 = [k]P_1$. In other words it means that torsion subgroups of prime order are isomorphic on isogenous curves. If n_1 is prime, then $k' = \frac{k}{\deg(\phi)} \bmod n_1$. It is important because in [1] there was proposition to count k' using formula $k' = \frac{k}{\deg(\phi)} \bmod \frac{\#E_2}{4}$. We will be looking for curves 2-isogenous over extension field over $GF(p^3)$. Curves over $GF(p^3)$ have much bigger order (about p^3) than curve over $GF(p)$ (about p). If k is odd using formula $k' = k/2 \bmod \#E_2/4$ (as it is proposed in [3]) results that the given number k' would be about three times longer than number k and thus scalar multiplication of point would be also about three times longer. If we use formula $k' = \frac{k}{\deg(\phi)} \bmod n_1$ then k' will be always about the same bitlength as k.

5 Isogenies between curves

In [3] and then also in [1] it is described how we can find 2-isogenous curves. If $p \equiv 1 (\bmod 4)$ every curve with three points of order 2 is birationally equivalent to a twisted Edwards curve. Unfortunately, this is not true if $p \equiv 3 (\bmod 4)$ if considered curve does not have any point of order 4. In both cases we are able to construct 2-isogenous twisted Edwards curve. The proof of this theorem may be found in [3]. Finally, to construct twisted Edwards curve to the given short Weierstrass curve we need to make transformations shown below:
Firstly we need to find roots of polynomial $x^3 + ax + b = (x - r_0)(x - r_1)(x - r_2)$. The transformation will be easier if one of the roots is equal 0. Let's suppose that:
$R_0 = 0$
$R_1 = r_1 - r_0$
$R_2 = r_2 - r_0$
Now we are able to construct curve E_2 which is isomorphic to curve E_1:
$E_2 : y^2 = x^3 - (R_1 + R_2)x^2 + R_1R_2x$ with point on this curve $P_2 = (x_2, y_2) = (x_1 - r_0, y_1)$
Now we are able to construct curve E_3 which is 2-isogenous to the elliptic curve E_2:
$E_3 : y^2 = x^3 + 2(R_1 + R_2)x^2 + (R_1 - R_2)^2x$ with point $P_3 = (x_3, y_3) = ((\frac{y_2^2}{x_2^2}), \frac{y_2(R_1R_2 - {x_2}^2)}{{x_2}^2})$

Now we will show the short proof of this fact:
Because $y^2 = x^3 - (R_1 + R_2)x^2 + R_1R_2x$ then $y^2 + (R_1 + R_2)x^2 = x^3 + R_1R_2x$

and $(y^2+(R_1+R_2)x^2)^2=(x^3+R_1R_2)^2x^2$. After that we get:
$y^4+2(R_1+R_2)y^2x^2+(R_1+R_2)^2x^4=(R_1^2R_2^2+2R_1R_2x^2+x^4)x^2$ and $y^4+2(R_1+R_2)y^2x^2+(R_1+R_2)^2x^4-4R_1R_2x^2=(R_1^2R_2^2-2R_1R_2x^2+x^4)x^2$. Now if we multiply both sides by $\frac{1}{x^4}$ we get:
$\frac{y^4}{x^4}+2(R_1+R_2)\frac{y^2}{x^2}+(R_1-R_2)^2=\frac{(R_1^2R_2^2-2R_1R_2x^2+x^4)}{x^2}$
and if we now multiply both sides by $\frac{y^2}{x^2}$:
$\frac{y^6}{x^6}+2(R_1+R_2)\frac{y^4}{x^4}+(R_1-R_2)^2\frac{y^2}{x^2}=\frac{y^2(R_1^2R_2^2-2R_1R_2x^2+x^4)}{x^4}$
And finally:
$(\frac{y^2}{x^2})^3+2(R_1+R_2)(\frac{y^2}{x^2})^2+(R_1-R_2)^2(\frac{y^2}{x^2})=(\frac{y(R_1R_2-x^2)}{x^2})^2$

It means that point $P_2=(x_2,y_2)$ on curve E_2 is mapped into point $P_3=((\frac{y_2^2}{x_2^2}),\frac{y_2(R_1R_2-{x_2}^2)}{{x_2}^2})$ on curve E_3. It is obvious that we cannot do this transformation for point $(0,0)$ which belongs to curve E_2. It means that point $(0,0)$ on E_2 will be mapped into point O on curve E_3. But we know that point O on curve E_2 is also mapped into point O on curve E_3. It means that points $O,(0,0)$ belongs to the kernel of isogeny and it means that it is 2-isogeny.

Now we may construct Montgomery curve E_4 which is isomorphic to curve E_3 by formula
$E_4:\frac{1}{R_1-R_2}y^2=x^3+\frac{2(R_1+R_2)}{R_1-R_2}x^2+x$ with point $P_4=(\frac{x_3}{R_1-R_2},\frac{y_2}{R_1-R_2})$

In [3] it is also proved that every Montgomery curve is birationally equivalent with twisted Edwards curve $E_5:4R_1x^2+y^2=1+4R_2x^2y^2$ with point $P_5=(\frac{x_4}{y_4},\frac{x_4-1}{x_4+1})$

If we want to use arithmetic on Edwards curve instead of arithmetic on twisted Edwards curve, we need to make one more transformation:

$E_6:x^2+y^2=1+4\frac{R_1}{R_2}x^2y^2$ with point $P_6=(x_5\sqrt{d},\frac{1}{y_5})$ if d is square in K or $E_6:x^2+y^2=1+4\frac{R_2}{R_1}x^2y^2$ with point $P_6=(x_5\sqrt{a},y_5)$ if a is square in K.

If both a and d are not squares in K then we need to use arithmetic on twisted Edwards curve.

5.1 Field extension

Elliptic curve has three points of order 2 if and only if equation:
$x^3+ax+b=0$ has three roots (y-coordinate of point of order 2 is always equal 0). If elliptic curve in short Weierstrass does not have three points of order 2 we need to consider curve with the same coefficients (in $GF(p)$) but with roots in $GF(p^3)$. Then we may be sure that there are always three roots of equation considered above. Because of this fact mapping from E_1 to E_5 will be considered for curves over $GF(p^3)$ (but coefficients of E_1 still belong to $GF(p)$). Then of course coefficients of E_5 belong to $GF(p^3)$ and point on $E_1:P_1=(x_1,y_1),x_1,y_1\in GF(p)$ is mapped into point on $E_5:P_5=(x_5,y_5),x_5,y_5\in GF(p^3)$. Of course if we count the point scalar multiplication on E_5 and map this point from E_5 to E_1 using dual isogeny ϕ', the given point on E_1 will be given by coordinates

$x, y \in GF(p)$. Unfortunately, all operations on E_5 must be done in $GF(p^3)$ which is harder than in $GF(p)$. However addition in $GF(p^3)$ is not much more complicated, unfortunately there are some problems to speed-up in $GF(p^3)$ multiplication and inversion. In software solutions it is very hard to speed up these operations but in FPGA, using parallel computations, it is much easier.

5.2 FPGA coprocessor for fast scalar multiplication

We create full FPGA implementation of coprocessor for fast scalar multiplication. Because we use twisted Edwards curve 2-isogenous to the short Weierstrass curve we need firstly to find out parameters of this curve and generator. Fortunately, for given curve we need to make such computations only once, so we can give these all parameters as constants. After that we need to count the value of $k \cdot 2^{-1} \bmod(r)$, where r is the order of torsion subgroup in which we operate (this is the same number as order of points group on short Weierstrass curve). Of course if k is even we can just shift the number k one place right (but we will not do that because it gives additional information during side channel attack). If k is odd we need to know the value of $2^{-1} \bmod(r)$. After these all preparations we are able to make the scalar multiplication of generator.
Of course the result we achieve will be the point on twisted Edwards curve over $GF(p^3)$. Moreover, we use the twisted inverted coordinates because they require the least multiplying operations. So at the end we need to transform the given point on twisted Edwards curve over $GF(p^3)$ into point on short Weierstrass curve over $GF(p^3)$ (but because r is prime we can assume that we get the point on short Weierstrass curve over $GF(p)$). The transformations are given by formulas:

If we use twisted Edwards curve arithmetic, we begin from
$E_5 : 4R_1x^2 + y^2 = 1 + 4R_2x^2y^2$ with point $P_5 = (x_5, y_5)$
Then the rest of operation is the same while using Edwards and twisted Edwards curve arithmetic:

$E_4 : \frac{1}{R_1-R_2}y^2 = x^3 + \frac{2(R_1+R_2)}{R_1-R_2}x^2 + x$ with point $P_4 = (\frac{1+y_5}{1-y_5}, \frac{1+y_5}{x_5(1-y_5)})$

$E_3 : y^2 = x^3 + 2(R_1 + R_2)x^2 + (R_1 - R_2)^2x$ with point $P_3 = (x_3, y_3) = (x_4(R_1 - R_2), y_4(R_1 - R_2))$

$E_2 : y^2 = x^3 - (R_1 + R_2)x^2 + R_1R_2x$ with point on this curve $P_2 = (x_2, y_2) = (\frac{{y_3}^2}{4{x_3}^2}, \frac{y_3(({R_1-R_2})^2-{x_3}^2)}{8{x_3}^3})$

$E_1 : y^2 = x^3 + ax + b$ with point on this curve $P_1 = (x_1, y_1) = (x_2 + r_0, y_2)$

Because it is always possible to use twisted Edwards curve arithmetic (to use Edwards curve arithmetic a or d must be non-square in K) we present transformations for this arithmetic.

It is easy to see that transformation from E_5 to E_1 requires 4 inversions, 9 multiplications and 13 additions/subtractions. Moreover, these computations must be done always at the end of point scalar multiplication.
Of course the biggest problem is too much inversions we need to count to get the point on E_1. If we analyze these all formulas more carefully, we will see that (now we consider that on curve E_5 we have the point in inverted twisted coordinates: $P_5 = (u_5, v_5, z_5)$):
$x_1 = \frac{{u_5}^2}{4{z_5}^2} + r_0$
$y_1 = \frac{u_5 v_5 (R_2 - R_1)}{2(z_5 - v_5)(z_5 + v_5)}$
Using simultaneous inversion we are able to count x_1, y_1 using only 1 inversion, 9 multiplications and 5 additions/subtractions.

We should remember that if we use Edwards curve arithmetic, at first we need to count:
If d is square in K we have:
$E_6 : x^2 + y^2 = 1 + 4\frac{R_1}{R_2}x^2y^2$ with point $P_6 = (x_6, y_6)$ and then we obtain
$E_5 : 4R_1x^2 + y^2 = 1 + 4R_2x^2y^2$ with point $P_5 = (x_5, y_5) = (\frac{x_6}{\sqrt{d}}, \frac{1}{y_6})$
or if a is square in K we have:
$E_6 : x^2 + y^2 = 1 + 4\frac{R2}{R1}x^2y^2$ with point $P_6 = (x_6, y_6)$ and then we obtain:

$E_5 : 4R_1x^2 + y^2 = 1 + 4R_2x^2y^2$ with point $P_5 = (x_5, y_5) = (\frac{x_6}{\sqrt{a}}, y_5)$.
Using Edwards curve arithmetic also need only 1 inversion and similar number

of multiplications and additions.

If $R' = R_2 - R_1$ where R_1 and R_2 are constant and we have the constant r_0, then:

1. $Z = 2z_5$
2. $A = z_5 - v_5$
3. $B = z_5 + v_5$
4. $C = A \cdot B$
5. $D = 2 \cdot C$
6. $E = u_5 \cdot v_5$
7. $F = E \cdot R'$
8. $G = F \cdot Z$
9. $H = D \cdot Z$
10. $H' = H^{-1}$
11. $y_1 = G \cdot H'$

12. $I = u_5 \cdot D$
13. $J = I \cdot H'$
14. $K = J^2$
15. $x_1 = K + r_0$

We are able to obtain similar formulas in the case when we use Edwards curve arithmetic. It will require additionally a few multiplications.

6 Multiplication in $GF(p^3)$

Operations in extended fields are computed by making operations modulo irreducible polynomial. The best are irreducible polynomials that have most of coefficient equal to 0. Then we look for polynomials that have most of coefficients equal to 1. Then for all non-zero and non-one coefficient we try find the smallest integer, for which the given polynomial is irreducible. We consider the polynomials of form $x^3 + x + c$. We can also consider polynomials of form $x^3 + ax^2 + 1, x^3 + bx + 1, x^3 + x^2 + c, x^3 + c$. The best for our purpose are polynomials of form $x^3 + c$, but unfortunately this form of irreducible polynomial does not exist for all NIST curves.

NIST curve	$x^3 + x + c$	$x^3 + c$
NIST P-192	$x^3 + x + 7$	Not exist
NIST P-224	$x^3 + x + 8$	$x^3 + 2$
NIST P-256	$x^3 + x + 13$	$x^3 + 2$
NIST P-384	$x^3 + x + 3$	Not exist
NIST P-521	$x^3 + x + 4$	$x^3 + 3$

Table 1: Irreducible polynomials of form $x^3 + x + c$ for NISTcurves.

The table shows number of additional processor cycles necessary for multiplication in $GF(p^3)$ comparing to multiplication in $GF(p)$ for irreducible polynomial $x^3 + x + c$ (using interleaved multiplier): for example for NIST P-192 multiplication in $GF(p^3)$ using irreducible polynomial of form $x^3 + x + 7$ require 198 processor cycles (199 if we count also initialization cycle) instead of 192 (193 if we count also initialization cycle) in $GF(p)$.

NIST curve	$x^3 + x + c$	$x^3 + c$
NIST P-192	+6	Not exist
NIST P-224	+6	+4
NIST P-256	+7	+4
NIST P-384	+5	Not exist
NIST P-521	+5	+5

Table 2: Number of additional processor cycles using multiplication in $GF(p^3)$ instead of multiplication in $GF(p)$.

If we use multiplication algorithm in $GF(p)$ by which we are able to make multiplication by number of 4 bits length in one processor cycle, then for irreducible polynomials we consider multiplication in $GF(p^3)$ will require 4 additional processor cycles comparing to multiplication in $GF(p)$.

In this paper we present the solution using polynomials of form $x^3 + x + c$ but we should always find the best for us solution just for given prime number p. The multiplication by constant need at most as many processor cycles as bits the number have. Sometimes (especially if the constant is power of 2) we may count the multiplication by this number with one processor cycle less.

6.1 Multiplication in $GF(p^3)$ using irreducible polynomial of form $F(x) = x^3 + x + c$

For all NIST curves over $GF(p)$ we found irreducible polynomials of form $F(x) = x^3 + x + c$. Using such polynomial we are able to multiply two elements $A = a_2x^2 + a_1x + a_0, B = b_2x^2 + b_1x + b_0$ where $A, B \in GF(p^3)$ using formula:
$A \cdot B = x^2(-M + L + U) + x(-c \cdot M - R + S) - c \cdot R + N$
Where:

1. $L = a_1 \cdot b_1$
2. $M = a_2 \cdot b_2$
3. $N = a_0 \cdot b_0$
4. $O = (a_1 + a_2)(b_1 + b_2)$
5. $P = (a_0 + a_1)(b_0 + b_1)$
6. $R = O - L - M$
7. $S = P - N - L$
8. $T = (a_0 + a_2)(b_0 + b_2)$
9. $U = T - M - N$
   These all operations require 6 multiplications in $GF(p)$, 2 multiplications by c which is small integer and 17 additions/subtractions in $GF(p)$

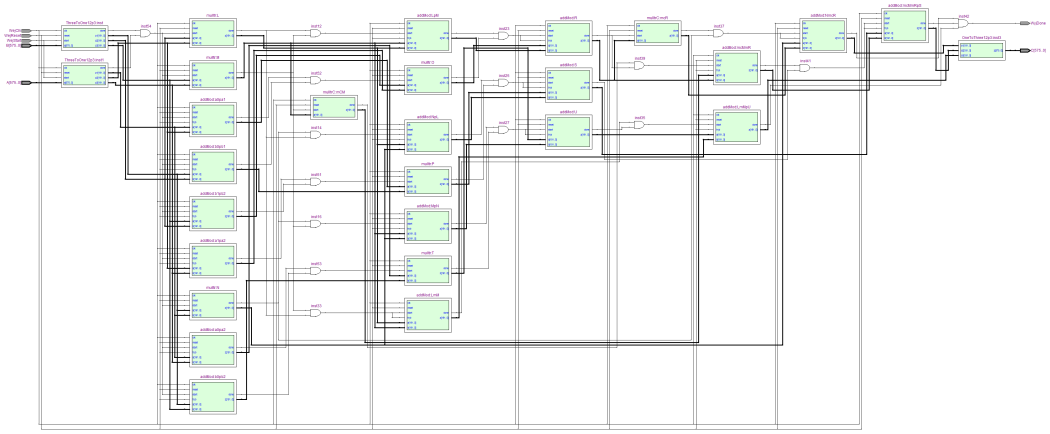

Figure 1: The scheme of FPGA implementation of multiplication in $GF(p^3)$.

7 Counting inversions in $GF(p^3)$

It is obvious that we are able to count for $A \in GF(p^3)$ its inversion A^{-1} using formula:
$A^{-1} = A^{p^3-2}$
We can use also extended Euclidean algorithm. In both cases counting inversion requires three times more steps than counting inversion for element from $GF(p)$. Moreover, every step requires making operations in $GF(p^3)$ instead of $GF(p)$ which are still a little bit slower.
It is possible for $A \in GF(p^3)$ to count its inversion A^{-1} by count one inversion of element from $GF(p)$. The method for irreducible polynomial of form $F(x) = x^3 + c$ may be found in [6]. We use very similar methods for irreducible polynomial $F(x) = x^3 + x + c$. Of course the same methods may be used for any other irreducible polynomials. Let's write:

$$A = \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix}, \text{ and } A^{-1} = \begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix}.$$

$$\text{If } M = \begin{bmatrix} (a_0 - a_2) & a_1 & a_2 \\ -(c \cdot a_2 + a_1) & (a_0 - a_2) & a_1 \\ -c \cdot a_1 & -c \cdot a_2 & a_0 \end{bmatrix} \text{ then}$$

$$M \cdot \begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} (a_0 - a_2) & a_1 & a_2 \\ -(c \cdot a_2 + a_1) & (a_0 - a_2) & a_1 \\ -c \cdot a_1 & -c \cdot a_2 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Coefficients in matrix M may be taken from general form of element $C = A \cdot B$.

Now we can transform it into:

$$\begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} (a_0 - a_2) & a_1 & a_2 \\ -(c \cdot a_2 + a_1) & (a_0 - a_2) & a_1 \\ -c \cdot a_1 & -c \cdot a_2 & a_0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = M^{-1} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Now we can do as follow: the determinant of matrix M is equal to:
$det(M) = 2a_2a_1c(a_0 - a_2) + (a_2c + a_1)(a_2{}^2c + a_1a_0) - a_1{}^3c + a_0(a_0 - a_2)^2$
Then:
$M^{-1} = \frac{1}{det(M)} \cdot$
$$\cdot \begin{bmatrix} a_1a_2 + a_0(a_0 - a_2) & -a_2^2 - a_0a_1 & a_1^2 - (a_0 - a_2)a_2 \\ a_0(a_2 \cdot c + a_1) - a_1^2 \cdot c & a_1a_2 \cdot c + a_0(a_0 - a_2) & -a_2(a_2 \cdot c + a_1) - a_1(a_0 - a_2) \\ a_1(a_0 - a_2) \cdot c + a_2 \cdot c(a_2 \cdot c + a_1) & (a_0 - a_2)a_2 zcdotc - a_1^2 \cdot c & (a_0 - a_2)^2 + a_1(a_2 \cdot c + a_1) \end{bmatrix}$$
And
$$\begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix} = \frac{1}{det(M)} \cdot \begin{bmatrix} a_1{}^2 - (a_0 - a_2)a_2 \\ -a_2(a_2c + a_1) - a_1(a_0 - a_2) \\ (a_0 - a_2)^2 + a_1(a_2 \cdot c + a_1) \end{bmatrix}$$
We can count these using formulas:

1. $A = a_0 - a_2$
2. $B = a_2 \cdot c$
3. $C = B + a_1$
4. $D = 2 \cdot a_1$
5. $E = D \cdot B$
6. $F = E \cdot A$
7. $G = a_2 \cdot B$
8. $H = a_1 \cdot a_0$
9. $I = G + H$
10. $J = C \cdot I$
11. $K = a_1^2$
12. $L = -K \cdot a_1$
13. $M = L \cdot c$
14. $N = A^2$
15. $P = N \cdot a_0$
16. $Q = F + J$
17. $R = M + P$

18. $S = Q + R$
19. $\hat{S} = S^{-1}$
20. $T = a_2 \cdot A$
21. $U = K - T$
22. $W = -a_1 \cdot A$
23. $X = -a_2 \cdot C$
24. $Y = W + X$
25. $Z = a_1 \cdot C$
26. $O = N + Z$
27. $b_2 = U\hat{S}$
28. $b_1 = Y\hat{S}$
29. $b_0 = O\hat{S}$

    These operations require 1 inversion, 18 multiplications and 10 additions/subtractions.

8 FPGA implementation of presented solution

8.1 Project assumptions

In literature we can find many fast solutions for point scalar multiplication in FPGA (see [10], [15], [16]). Unfortunately used techniques strongly depends on the field size, used type of multiplication, inversion and scalar multiplication method (binary, NAF, wNAF etc.). Moreover, the best solution for device invulnerable for side channel attack may be constructed using much different techniques than the best solutions for device vulnerable for side channel attack. Because our main aim was to show idea of using (twisted) Edwards curve over $GF(p^3)$ to count point scalar multiplication for NIST curves in FPGA, we made assumptions:

- multiplication is counted using interleaved multiplier (multiplication of two $L - bit$ numbers requires $L + 1$ processor cycles - L cycles for multiplication + one initialization cycle)

- inversion is counted using fast exponentiation, because we need to count it only once and other algorithms (for example extended Euclidean algorithm) require much more components which means that we would require much more resources.

- point scalar multiplication is counted using binary (double-and-add) method

- every operation that require access to the RAM memory require one more processor cycle for initialization

Of course devices made using these assumptions are not the fastest, but they clearly show the presented ideas. The presented ideas may be used to get better solution using different project assumptions but it is not the main objective of presented work.

One of the most important things to speed up the scalar multiplication is choosing multiplication algorithm. We chose interleaved multiplier to show advantages of using twisted Edwards curves over $GF(p^3)$. In other cases we should careful choose how to implement point scalar multiplication and what multiplication algorithm in $GF(p)$ would be the best. If multiplication algorithm in $GF(p)$ requires at least 30-40 processor cycles then using twisted Edwards curves over $GF(p^3)$ for point scalar multiplication may be very good idea.

8.2 Fast implementation of multiplication in $GF(p^3)$ using FPGA

It is easy to see that multiplication on $GF(p^3)$ requires much more operations than multiplication in $GF(p)$. In software solutions this fact makes using twisted Edwards curves over $GF(p^3)$ absolutely useless. The things look different if we consider hardware solutions, for example FPGA. In FPGA we are able to make parallel computations which decrease time needed to make multiplication in $GF(p^3)$. Moreover, because points addition and point doubling on twisted Edwards curve in inverted coordinates is much faster than the same operations on short Weierstrass curve, in some cases the time required for scalar point multiplication may be much shorter using twisted Edwards curves over $GF(p^3)$ than short Weierstrass curves over $GF(p)$. The scheme of multiplication is shown below:

8.3 FPGA coprocessor

In this article we present a FPGA implementation of working coprocessor for fast scalar multiplication using twisted Edwards curves. We need to get in result point $Q = [k]P$. Scalar multiplication is divided into three main steps:

1. Counting $k'^{-1} mod(r)$, where r is the order of generator G (G is point on short Weierstrass curve and is mapped by 2-isogeny ϕ into point G_t which is generator of subgroup of order r on (twisted) Edwards curve).

2. Counting $Q_t = [k']G_t$, where Q_t is the point on twisted Edwards curve in inverted coordinates e element

3. Transformation from Q_t to Q (including counting inversion)

The first step is counted using component for multiplication in $GF(p)$ which on the scheme is described $Mul\_GF(p)$. This device has two modes: the first mode is used to count $a \cdot b \bmod(p)$, the second mode to count $a \cdot b \bmod(r)$. So if we want to count $k'^{-1} mod(r)$ the $Mul\_GF(p)$ device must be set to mode 1.

In other case (especially when we count inversion of element in $GF(p^3)$ using inversion in $GF(p)$) the mode must be set for 0.

The second step is counted using components: RAM_L, $ADD\_GF(p\hat{}3)$, $MUL\_GF(p\hat{}3)$, where L is the bitlength of p.

RAM_3L is RAM memory consisted of 14 registers of $3L$ bitlength each. This is necessary because every element from $GF(p^3)$ for p which is L bits length consists from three elements each of L bitlength. The element $A = a_2x^2 + a_1x + a_0$ is written in memory as concatenation of coefficients: $a_2|a_1|a_0$.

Component $ADD\_GF(p\hat{}3)$ is used for addition or subtraction of elements from $GF(p^3)$. If the mode is set on 0 then component is used for addition, if mode is set on 1 then component is used for subtraction. Component $MUL\_GF(p\hat{}3)$ is used for multiplication of elements from $GF(p^3)$. It is the most complex element of all components.

Transformation of point from E_5 to E_1 is made by using almost all components because most of operations are made for elements from $GF(p^3)$ but for counting inversion we use components RAM_L, $ADD\_GF(p)$, $MUL\_GF(p)$ which all are used for making operations in $GF(p)$.

However, the presented solution requires more steps than traditional scalar multiplication, the fact of using faster arithmetic on twisted Edwards curves results in that the time of presented solution is shorter.

Compilation results for STRATIX IV using twisted Edwards curve arithmetic (because differences between using twisted Edwards curve arithmetic and Edwards curve arithmetic are very small then the hardware requirements are almost the same and we will not present this case):

NIST curve	#ALUTs	#REGISTERS	#RAM	#PINs	Max frequency [MHz]
NIST P-192	27559	12398	21504	165	85.11
NIST P-224	31026	13389	25088	165	81.16
NIST P-256	33777	15206	28672	165	78.44
NIST P-384	53469	22301	43008	165	59.14
NIST P-521*	72545	30257	58352	165	40.84

Table 3: Compilation results for STRATIX IV using twisted Edwards curve arithmetic.

*For NIST P-521 compilation was not successful on STRATIX IV device. The values for this case have been approximated.

We made comparison of given solution in two different ways. Firstly we compared solutions which are vulnerable for side channel attack and we use in both cases: for short Weierstrass curve and (twisted) Edwards curves different formulas for point addition and point doubling (we use not unified formulas).

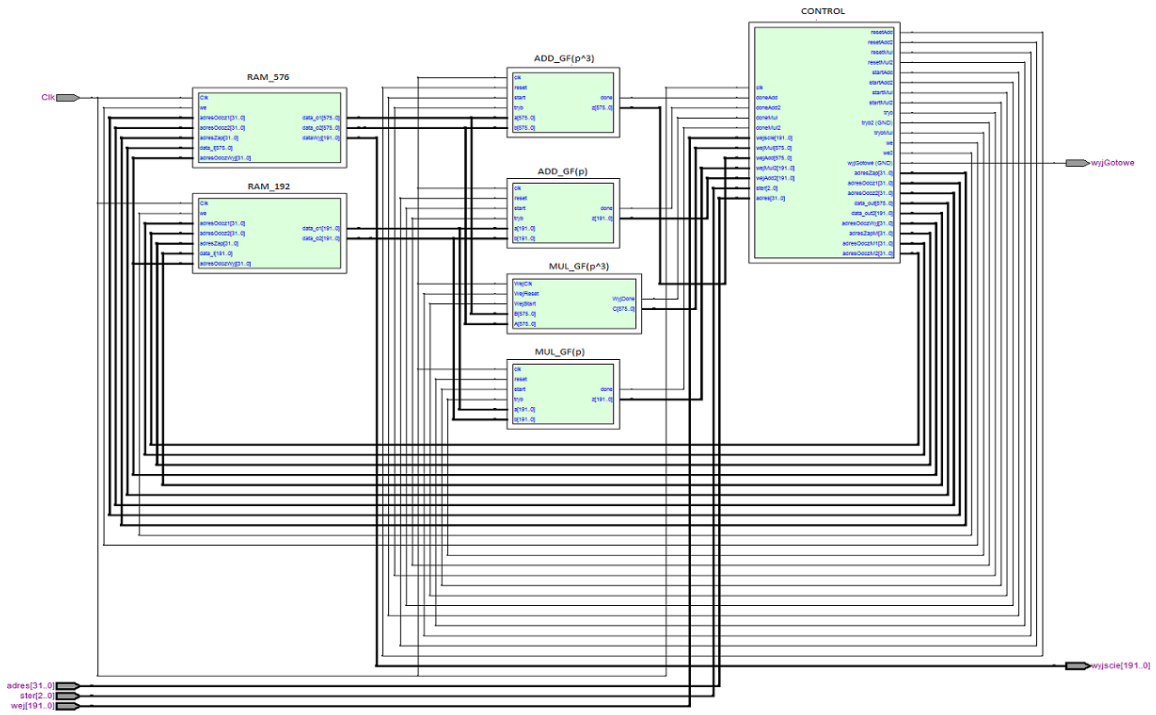

Figure 2: The scheme of FPGA implementation of coprocessor using presented techniques.

Then we compared solution which should be invulnerable for side channel attack. We made assumptions that for short Weierstrass curve we use Montgomery ladder and for (twisted) Edwards curve we use unified formula.

Number of processor cycles (including additional and initialization cycles) using short Weierstrass curve arithmetic (SWC) and using twisted Edwards curve arithmetic (TEC):

NIST curve	SWC $GF(p)$	TEC $GF(p^3)$ not unified	$\frac{TEC\ GF(p^3)\ not\ unified}{SWC\ GF(p)}$ [%]
NIST P-192	728930	636536	87.32
NIST P-224	975794	846472	86.75
NIST P-256	1258498	1089837	86.60
NIST P-384	2747714	2336003	85.02
NIST P-521	4976978	4204204	84.47

Table 4: Number of processor cycles for devices vulnerable for side channel attacks (for TEC).

NIST curve	SWC $GF(p)$ Ladder	TEC $GF(p^3)$ unified	$\frac{TEC\ GF(p^3)\ unified}{SWC\ GF(p)\ ladder}$ [%]
NIST P-192	934082	770648	82.50
NIST P-224	1254562	1028024	81.94
NIST P-256	1622146	1326893	81.80
NIST P-384	3563522	2860931	80.28
NIST P-521	6476416	5166230	79.77

Table 5: Number of processor cycles for device unvulnerable for side channel attacks (for TEC).

Number of processor cycles (including additional and initialization cycles) using Edwards curve arithmetic (EdC):

NIST curve	SWC $GF(p)$	EdC $GF(p^3)$ not unified	$\frac{EdC\ GF(p^3)\ not\ unified}{SWC\ GF(p)}$ [%]
NIST P-192	728930	579224	79.46
NIST P-224*	-	-	-
NIST P-256	1258498	988461	78.54
NIST P-384	2747714	2111363	76.84
NIST P-521	4976978	3792353	76.20

Table 6: Number of processor cycles for devices vulnerable for side channel attacks (for EdC).

NIST curve	SWC $GF(p)$ Ladder	EdC $GF(p^3)$ unified	$\frac{EdC\ GF(p^3)\ unified}{SWC\ GF(p)\ ladder}$ [%]
NIST P-192	934082	713336	76.37
NIST P-224*	-	-	-
NIST P-256	1622146	1225517	75.55
NIST P-384	3563522	2636291	73.98
NIST P-521	6476416	4754380	73.41

Table 7: Number of processor cycles for devices unvulnerable for side channel attacks (for EdC).

*For NIST P-224 we cannot use Edwards curve arithmetic using our solution because a and d are non-squares in K.

We compared the FPGA implementations of coprocessor for scalar multiplication on twisted Edwards curve over $GF(p^3)$ using presented techniques to coprocessor for scalar multiplication on short Weierstrass curve over $GF(p)$, which uses similar techniques and components.

The table below shows comparison:

Curve	Field	Max frequency [MHz]
TEC	NIST P-192	78.04
SWC	NIST P-192	82.71

Table 8: Maximal processor frequency for devices using twisted Edwards curves and short Weierstrass curves.

9 Protection against side channel attacks

It is well known that using Montgomery ladder protects against most types of side channel attacks. It is also easy to see that using unified formula brings similar protection against side channel attacks. If we use Montgomery ladder to count $Q = [k]P$ then for fields of bitlength L we must always count L additions of points and L point doublings. So it is impossible to guess any information about the number k.

If we use unified formula, doubling and addition are indistinguishable. The only information we can get, is (for example if we use binary method for scalar multiplication) the Hamming weight of the number k which we denote as $W(k)$. Using this information we can see that:

Probability that $W(k) = m$ is $\frac{\binom{L}{m}}{2^L}$. Even if know the number $W(k) = m$, then the expected number of possibilities we need to check is $\frac{\binom{L}{m}}{2}$. So for random number k the expected number of possibilities we need to check is:

$$U(L) = \sum_{i=0}^{L} \frac{\binom{L}{i}^2}{2^{L+1}} = \frac{\sum_{i=0}^{L} \binom{L}{i}^2}{2^{L+1}} = \frac{\binom{2L}{L}}{2^{L+1}}$$

In the case of using Montgomery ladder the expected number of possibilities we need to check is $M(L) = 2^{L-1}$. So we can see that using Montgomery ladder is a little bit save than using unified formula, because $\frac{\binom{2L}{L}}{2^{L+1}} \leq 2^{k-1}$ and $\frac{U(L)}{M(L)} = \frac{\binom{2L}{L}}{2^{2L}}$

NIST curve	L	$\frac{U(L)}{M(L)} \cdot 100\%$
NIST P-192	192	4.07
NIST P-224	224	3.77
NIST P-256	256	3.52
NIST P-384	384	2.88
NIST P-521	521	2.47

Table 9: Comparison of security Montgomery ladder and unified formula.

So we can see that using unified formula instead of Montgomery ladder in the case of side channel attack is less save at most for only from $\lceil \log_2 \frac{100}{4.07} \rceil = 5$ bits (when we use NIST P-192) to $\lceil \log_2 \frac{100}{2.47} \rceil = 6$ bits (when we use NIST P-521). Knowing Hamming weight of number k for which we make scalar multiplication does not decrease the security of device so much in the case of side channel attack.

10 Conclusion

Presented solution is much faster than traditional point scalar multiplication in $GF(p)$ when multiplication requires many processor cycles (at least 30-40) which is very frequent in hardware implementations. Unfortunately the presented device requires much more logic elements than traditional solutions. Because of that we always need to choose method for point scalar multiplication very carefully.

10.1 Improvements and advantages

In this work we made some improvements due to previous articles about this topic:
- We made first FPGA device for point scalar multiplication using unified formula which protect this device against side channel attacks
- The presented solution uses fast operations on (twisted) Edwards curve over $GF(p^3)$ which we can use to speed-up point scalar multiplication for NIST curves in some cases.
- Presented solution is faster than classic solutions up to 23% if we consider device vulnerable for side channel attacks and up to 26% if we consider device invulnerable for side channel attacks
- For all NIST curves we found irreducible polynomials of special form $F(x) =$

$x^3 + x + c$ to increase the efficiency of operations in $GF(p^3)$.
- We showed that if a or d is square in K then we can find Edwards curve birationally equivalent to the given twisted Edwards curve and thus we can use Edwards curve arithmetic which require less multiplications than twisted Edwards curve arithmetic. Moreover, in this case we can count point scalar multiplication for all points of odd order using field extension $GF(p^3)$ and it is not necessary to use field extension $GF(p^6)$ as it was proposed in [1].
- We present full working device for point scalar multiplication for NIST curves using 2-isogenous twisted Edwards curves. The longer the single multiplication is, the more time we are able to save.

10.2 Disadvantages

Unfortunately, this solution has some disadvantages and limits. The main disadvantages are:
- The presented solution should be considered only if multiplication requires more than 30-40 processor cycles (average case)
- The FPGA structure of presented solution requires about three to four times more elements than traditional solutions
- The clock speed is a little bit slower because of using operations in $GF(p^3)$ instead of using operations in $GF(p)$
- If field is big number (for example NIST P-521) the solution for such field requires too many resources to use it in practice.

11 Further work

In our opinion we should consider to use standard short Weierstrass curves or Edwards or twisted Edwards curves defined over $GF(p^3)$ with prime order (for short Weierstrass curve) or order equal to $4q$, where q is prime number. Using such curves we would be able to count multiplication and inversion much faster than now and because order of points group would be prime number close to p^3, then the security of this solution would be similar to the security while using curve over $GF(t)$, where t is three times longer (in bits) prime than p. Using Edwards curves arithmetic we would be able to get in hardware solutions from 3 times faster (if we need not protection against side channel attacks) to even more than 4 times faster (if we need protection against side channel attacks). This is the result that multiplication and inversion in $GF(p^3)$ (which is needed only if we want to get point in affine coordinates) would be about three times faster than in $GF(t)$ and additionally using Edwards curve arithmetic require less multiplications. We are also able to count inversion much faster, because inversion in $GF(p^3)$ may be counted using one inversion in $GF(p)$ and some additional multiplications and additions/subtractions. Moreover, the presented in this article FPGA implementation of coprocessor may be used to that after few small changes. Of course there are known some index calculus methods for

curves over field extension and we should study all advantages and disadvantages of every possible solution very carefully.

References

[1] V. Verneuil. "Elliptic Curve Cryptography on Standard Curves Using the Edwards Addition Law." Yet Another Conference on Cryptography. 2008.

[2] „http://www.hyperelliptic.org/." .

[3] P. Birkner, M. Joye, T. Lange, Ch. Peters, D. Bernstein. "Twisted Edwards Curves." eprint.iacr.org (2008).

[4] T. Lange, D. Bernstein. "Faster addition and doubling on elliptic curves." Cryptology ePrint Archive, 2007.

[5] D. Bernstein, P. Birkner, T. Lange, Ch. Peters. "ECM using Edwards curves." Cryptology ePrint Archive, 2008.

[6] H. Cohen, G. Frey. "Handbook of Elliptic and Hyperelliptic Curve Cryptography." New York: Chapman & Hall/CRC, 2006.

[7] I. Blake, G. Seroussi, N. Smart. "Elliptic curves in cryptography". Cambridge University Press, 1999

[8] H. Hisil, K. Koon-Ho Wong, G. Carter, E. Dawson. "Twisted Edwards Curves Revisited". ASIACRYPT 2008

[9] J. Silverman. The arithmetic of elliptic curves, Second edition. Nowy Jork: Springer, 2009.

[10] N. Guillermin. „A high speed coprocessor for elliptic curve scalar multiplications over Fp."

[11] D. Genkin, A. Shamir, E. Tromer. „RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis." Cryptology ePrint Archive 2013, wyd. Report 2013/857.

[12] B. Smith. Mappings of elliptic curves. Eindhoven: INRIA, 2008.

[13] P. Kocher, J. Jaffe, B. Jun. „Differential power analysis." Advances in Cryptology - Crypto 99 Proceedings. Lecture Notes In Computer Science 1999.

[14] M. Neunhöffer. Module MT 5826 Finite Fields. RWTH Aachen University. Aachen, 2007.

[15] Yu-Shiang WANG, Chih-Tsun HUANG Jyu-Yuan LAI. „High-Performance Architecture for Elliptic Curve Cryptography over Prime Fields on FPGAs." Interdisciplinary Information Sciences 2012.

[16] Zongbin Liu, Wuqiong Pan, Jiwu Jing Yuan Ma. „A High-Speed Elliptic Curve Cryptographic Processor for Generic Curves over GF(p).” 2013.