

АЛГОРИТМЫ СКАЛЯРНОГО УМНОЖЕНИЯ В ГРУППЕ ТОЧЕК ЭЛЛИПТИЧЕСКОЙ КРИВОЙ И НЕКОТОРЫЕ ИХ МОДИФИКАЦИИ

Введение

Теория алгебраических кривых, и в частности преобразования в группе точек эллиптической кривой, сегодня стали основой построения ряда новых криптографических стандартов [1, 7, 8, 36, 37]. Интерес к этому математическому аппарату обусловлен открывшимися возможностями уменьшения, по сравнению с другими стандартами эквивалентной стойкости, длины ключа и блока, на котором выполняются криптопреобразования.

Одним из актуальных направлений развития современных криптографических преобразований в группе точек эллиптической кривой является уменьшение сложности выполнения базовых операций и как следствие скалярного умножения точек эллиптической кривой. Задачи уменьшения сложности привлекли внимание многих специалистов, в результате исследований и разработки которых созданы средства выполнения операций скалярного умножения с повышенной скоростью. В статье ставится цель изложить сущность и особенности различных алгоритмов уменьшения сложности скалярного умножения, которые удалось выделить из доступных публикаций, выполнить их систематизацию по вычислительной сложности и определить наиболее перспективные направления оптимизации. Представленные материалы вместе с обширной библиографией, на наш взгляд, могут стать полезными для специалистов-практиков, занимающихся разработкой средств криптографической защиты информации на основе преобразований в группе точек эллиптических кривых, якобиане дивизоров гиперэллиптических кривых, а также специалистов криптоаналитиков, успешное решение задач криптоанализа которыми в существенной мере зависит от сложности выполнения скалярного умножения.

В статью также включены предложения авторов по дополнительному улучшению некоторых алгоритмов скалярного умножения, появившиеся в процессе ее подготовки.

1. Алгоритмы скалярного умножения

Сразу отметим, что речь будет идти об операциях с точками эллиптической кривой (ЭК) рекомендованных в [37], которые рассматриваются над полями характеристики 2.

Операция скалярного умножения (СУ) состоит в вычислении точки $Q = kP$, где $P \in E(GF(2^m))$ – точка ЭК, имеющая известный порядок n , k – целое, $k \in [1, n-1]$ [1, 5, 7, 8].

Приведем сначала алгоритм умножения, следующий из общего определения СУ точек ЭК.

Алгоритм 1. Двоичное умножение слева направо

Вход: $k = (k_{t-1}, \dots, k_0)_2$, $k_{t-1} = 1$, $P \in E(GF(2^m))$

Выход: kP

1. $Q \leftarrow O$.
 2. For $i = t-1$ downto 0 do
 - 2.1. $Q \leftarrow 2 \cdot Q$.
 - 2.2. If $k_i = 1$ then $Q \leftarrow Q + P$.
 3. Return Q .
-

В этом алгоритме, количество единиц в двоичном представлении скалярного множителя k , в среднем, составляет $t/2 \approx m/2$, что соответствует количеству операций сложения точек, в то время как количество операций удвоения равно m (см. алгоритм 1). Поэтому сложность алгоритма 1, в групповых операциях, составляет:

$$I(A_1^{aver}) = \frac{m}{2} I_{add} + m I_{dbl}. \quad (1)$$

Дальнейшая конкретизация этого выражения связана с выбором базиса представления точек (аффинного, проективного, смешанного). Например, в аффинных координатах точек, сложность алгоритма 1 в полевых операциях составит [1]:

$$I(A_1^{affin}) = 3m I_{mul} + \frac{3m}{2} I_{inv}, \quad (2)$$

Максимальной производительности групповых операций позволяет добиться представления точек ЭК в смешанном базисе [6]. Если в алгоритме 1, представить точку Q в модифицированных проективных координатах Лопеса-Дахаба, а точку P в аффинных – удвоение на шаге 2.1. и сложение на шаге 2.2. может быть выполнено с использованием выражений из [6]:

$$\begin{aligned} X_3 &= A^2 + K + L + a \cdot Z_3, \\ Y_3 &= X_3 \cdot (K + Z_3) + L \cdot C \cdot (F \cdot I + E \cdot J), \\ Z_3 &= D^2, \end{aligned}$$

где $E = Y_1 \cdot Z_2^2$, $F = Y_2 \cdot Z_1^2$, $A = F + E$, $I = X_1 \cdot Z_2$, $J = X_2 \cdot Z_1$, $B = I + J$, $C = Z_1 \cdot Z_2$, $D = B \cdot C$, $K = A \cdot D$, $L = B^2 \cdot D$.

В этом случае для оценки сложности алгоритма 1 в полевых операциях можно получить выражения:

$$I(A_1^{prj}) = (5m + 5m + 2) I_{mul} + 1 I_{inv} = (10m + 2) I_{mul} + 1 I_{inv}. \quad (3)$$

При получении этого выражения учтено, что сложность перехода из проективного представления Лопеса-Дахаба $(X : Y : Z)$ к аффинному $(X/Z, Y/Z^2)$ равна $I(T) = 2I_{mul} + 1I_{inv}$.

Очередной возможностью в повышении эффективности выполнения операции СУ предлагается учет того факта, что вычитание точек ЭК над двоичным полем эффективнее сложения (точке $P(x, y) \in E(GF(2^m))$ соответствует точка $-P = (x, x + y)$). Использование этого свойства требует изменения представление скалярного множителя k с двоичного на знаковое в несмежной форме (NAF – non-adjacent form) $NAF(k) = \sum_{i=0}^{l-1} k_i 2^i$, где $k_i \in \{0, \pm 1\}$. Полезность NAF [3, 5, 20, 28] состоит в том, что в нем смежные коэффициенты не могут быть одновременно отличными от нуля. Кроме того, NAF имеет наименьшее количество коэффициентов $k_i \neq 0$ среди других знаковых представлений и $NAF(k)$ длиннее двоичного представления числа k не более, чем на 1 [31, 32]. Каждое положительное целое k имеет единственное $NAF(k)$ представление. Эффективное вычисление $NAF(k)$ выполняется согласно алгоритма 2 [3, 5, 20, 28].

Алгоритм 2. Вычисление NAF целого числа

Вход: Целое число k

Выход: $NAF(k)$

1. $i \leftarrow 0$.

2. While $k \geq 1$ do

2.1. If $k \bmod 2 = 1$ then $k_i \leftarrow 2 - (k \bmod 4)$, $k \leftarrow k - k_i$ Else $k_i \leftarrow 0$.

2.2. $k \leftarrow k/2, i \leftarrow i+1$.

3. Return (k_{i-1}, \dots, k_0) .

Модифицированный алгоритм 3 с учетом использования $\text{NAF}(k)$ имеет вид [3]:

Алгоритм 3. Двоичное NAF умножение

Вход: $\text{NAF}(k) = \sum_{i=0}^{l-1} k_i 2^i, P \in E(GF(2^m))$

Выход: kP

1. $Q \leftarrow O$.

2. For $i = l-1$ downto 0 do

2.1. $Q \leftarrow 2 \cdot Q$.

2.2. If $k_i = 1$ then $Q \leftarrow Q + P$.

2.3. If $k_i = -1$ then $Q \leftarrow Q - P$.

3. Return Q .

Сложность алгоритма 3 с учетом того, что средняя плотность всех ненулевых коэффициентов в $\text{NAF}(k)$ длины m равна $\frac{m}{3}$ [3, 20], составит:

$$I(A_3) = \frac{m}{3} I_{add} + m I_{dbl}. \quad (4)$$

В случае реализации алгоритма без ограничения памяти, время его работы можно сократить посредством использования, так называемого, «оконного» метода предложенного в [3], суть которого состоит в следующем. На каждом шаге итерации анализируется w бит числа k за итерацию. При этом множитель k в $\text{NAF}_w(k)$ представляется в виде $\text{NAF}_w(k) = \sum_{i=0}^{l-1} k_i 2^i$, где каждый ненулевой коэффициент k_i – нечетный, $k_i < 2^{w-1}$, в остальных свойствах NAF и $\text{NAF}_w(k)$ аналогичны [3]. Заметим, что $\text{NAF}_2(k) = \text{NAF}(k)$. $\text{NAF}_w(k)$ вычисляется по алгоритму 4 [3, 5].

Алгоритм 4. Вычисление $\text{NAF}_w(k)$ целого числа

Вход: Целое число k

Выход: $\text{NAF}_w(k)$

1. $i \leftarrow 0$.

2. While $k \geq 1$ do

2.1. If $k \bmod 2 = 1$ then $k_i \leftarrow (k \bmod 2^w), k \leftarrow k - k_i$ Else $k_i \leftarrow 0$.

2.2. $k \leftarrow k/2, i \leftarrow i+1$.

3. Return (k_{i-1}, \dots, k_0) .

Согласно рекомендациям [3] ширина «окна» предвычислений для СУ в проективных координатах $w=4$ и $w=5$ для $m=163$ и $m=233$ ($m=283$) соответственно позволяет достичь минимальной сложности СУ. В дальнейшем принимается $w=4$, т.к. в этом случае сложность незначительно выше, чем при $w=5$, и значительно проще программная реализация.

В работах [3, 5, 20] предложен еще один алгоритм 5, использующий NAF представление и «окно» предвычислений, реализация которого представлена ниже:

Алгоритм 5. «Оконное» NAF умножение

Вход: $\text{NAF}(k) = \sum_{i=0}^{l-1} k_i 2^i, P \in E(GF(2^m))$

Выход: kP

1. Вычислить $P_i = iP$, для $i \in \{1, 3, 5, \dots, 2^{w-1} - 1\}$.

2. $Q \leftarrow O$.
3. For $i = l - 1$ downto 0 do
 - 3.1. $Q \leftarrow 2Q$.
 - 3.2. If $k_i \neq 1$ then
 - If $k_i > 0$ then $Q \leftarrow Q + P_{k_i}$.
 - Else $Q \leftarrow Q - P_{k_i}$.
3. Return (Q) .

Средняя плотность ненулевых коэффициентов в представлении $\text{NAF}_w(k)$ длиной m равна $m/(w+1)$ [3, 5, 20]. Для оценки сложности алгоритма 5 получено выражение:

$$I(A_5) = (1I_{dbl} + (2^{w-2} - 1)I_{add}) + (m/(w+1)I_{add} + mI_{dbl}). \quad (5)$$

Частным случаем, описанного выше алгоритма, для $w = 2$, следует считать алгоритм СУ в проективных координатах Якоби представленный в стандартах [2, 4, 5]:

Алгоритм 6. Умножение в проективных координатах Якоби со сложением и вычитанием
Вход: $k = (k_{t-1}, \dots, k_1, k_0)_2$, $k_{t-1} = 1$, $3k = (h_{t-1}, \dots, h_0)_2$, $P = (x, y) \in E(GF(2^m))$
Выход: kP

1. If $(k = 0)$ or $(Z = 0)$ then Return $(1, 1, 0)$.
2. $e = k \bmod n$, где n – порядок точки P . (если n - неизвестно, тогда $e = k$).
3. $X' \leftarrow X$, $Z' \leftarrow Z$, $Z_1 \leftarrow 1$.
4. If $(k < 0)$ then $k \leftarrow (-k)$, $Y' \leftarrow XZ + Y$.
5. else $Y' \leftarrow Y$.
6. If $Z' = 1$ then $X_1 \leftarrow X'$, $Y_1 \leftarrow Y'$.
7. else $X_1 \leftarrow X'/(Z')^2$, $Y_1 \leftarrow Y'/(Z')^3$.
8. For $i = t - 1$ downto 1 do
 - 8.1 $(X', Y', Z') \leftarrow 2(X', Y', Z')$.
 - 8.2. If $(h_i = 1)$ and $(k_i = 0)$ then $(X', Y', Z') \leftarrow (X', Y', Z') + (X_1, Y_1, Z_1)$.
 - 8.3. If $(h_i = 0)$ and $k_i = 1$ then $(X', Y', Z') \leftarrow (X', Y', Z') - (X_1, Y_1, Z_1)$.
9. Return (X', Y', Z') .

В основе следующего алгоритма лежит идея Монтгомери, предложенная в работе [13]. Ее сущность состоит в том, что x – координата точки $Q_1(x_1, y_1) + Q_2(x_2, y_2)$, $Q_1 \neq \pm Q_2$ может быть получена из x – координат точек $Q_1, Q_2, Q_1 - Q_2$ в соответствии с алгоритмом:

Алгоритм 7. Двоичное умножение Монтгомери
Вход: $k = (k_{t-1}, \dots, k_1, k_0)_2$, $k_{t-1} = 1$, $P = (x, y) \in E(GF(2^m))$
Выход: kP

1. $P_1 \leftarrow P$, $P_2 \leftarrow 2P$.
2. For $i = t - 1$ downto 0 do
 - 2.1. If $k_i = 1$ then $P_1 \leftarrow P_1 + P_2$, $P_2 \leftarrow 2P_2$.
 - 2.2. else $P_2 \leftarrow P_2 + P_1$, $P_1 \leftarrow 2P_1$.
3. Return (P_1) .

Сложность алгоритма 7 составит:

$$I(A_7) = (m-1)I_{add} + (m-1)I_{dbl}. \quad (6)$$

Модифицированный алгоритм СУ получается путем перехода в алгоритме 7 к операциям над точками $Q_1(x_1, y_1)$, $Q_2(x_2, y_2)$, $Q_1 \neq \pm Q_2$ в аффинных координатах. Если обозначить $Q_1 + Q_2 = (x_3, y_3)$ и $Q_1 - Q_2 = (x_4, y_4)$, то формула для вычисления суммы точек, принимает вид:

$$x_3 = x_4 + \frac{x_1}{x_1 + x_2} + \left(\frac{x_1}{x_1 + x_2} \right)^2. \quad (7)$$

На j -ой итерация алгоритма вычисляется $T_j = (k_j P, (k_j + 1)P)$. Затем, если старший коэффициент k_{j+1} равен 0, то $T_{j+1} = (2k_{j+1}P, (2k_{j+1} + 1)P)$ иначе $T_{j+1} = ((2k_{j+1} + 1)P, (2k_{j+1} + 2)P)$. Координаты результирующей точки $kP = (x_1, y_1)$ определяется как:

$$y_1 = x^{-1}(x_1 + x)[(x_1 + x)(x_2 + x) + x^2 + y] + y.$$

Представленные преобразования, использующие аффинное представление точек ЭК, реализованы в алгоритме 8.

Алгоритм 8. Умножение Монтгомери в аффинных координатах

Вход: $k = (k_{t-1}, \dots, k_1, k_0)_2$, $k_{t-1} = 1$, $P = (x, y) \in E(GF(2^m))$

Выход: kP

-
1. $x_1 \leftarrow x, x_2 \leftarrow x^2 + b/x^2$.
 2. For $i = t - 2$ downto 0 do
 - 2.1. $t \leftarrow x_1 / (x_1 + x_2)$.
 - 2.2. If $k_i = 1$ then
 - 2.2.1. $x_2 \leftarrow x + t^2 + t, x_2 \leftarrow x_2^2 + b/x_2^2$.
 - 2.3. else
 - 2.3.1. $x_2 \leftarrow x + t^2 + t, x_1 \leftarrow x_1^2 + b/x_1^2$.
 3. $r_1 \leftarrow x_1 + x, r_2 \leftarrow x_2 + x, y_1 \leftarrow r_1(r_1 r_2 + x^2 + y) / (x + y)$.
 4. Return (x_1, y_1) .
-

Сложность алгоритма 8, по нашим оценкам, составляет:

$$I(A_8) = (t - 2)(2I_{mul} + 2I_{inv} + 2I_{sqr}) + (1I_{inv} + 1I_{mul} + 1I_{sqr}). \quad (8)$$

В алгоритме 9, изложенном в работе [3, 11], авторы использовали идею Монтгомери, но реализовали операции над точками ЭК в стандартных проективных координатах.

Алгоритм 9. Умножение Монтгомери в стандартных проективных координатах

Вход: $k = (k_{t-1}, \dots, k_1, k_0)_2$, $k_{t-1} = 1$, $P = (x, y) \in E(GF(2^m))$

Выход: kP

-
1. Вычисление $(P, 2P)$: $X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2$.
 2. For $i = t - 2$ downto 0 do
 - 2.1. If $k_i = 1$ then
 - 2.1.1. $T \leftarrow Z_1, Z_1 \leftarrow (X_1 Z_2 + X_2 Z_1)^2, X_1 \leftarrow x Z_1 + X_1 X_2 T Z_2$.
 - 2.1.2. $T \leftarrow X_2, X_2 \leftarrow X_2^4 + b Z_2^4, Z_2 \leftarrow T^2 Z_2^2$.

2.2. else

$$2.2.1. T \leftarrow Z_2, Z_2 \leftarrow (X_1Z_2 + X_2Z_1)^2, X_2 \leftarrow xZ_2 + X_1X_2TZ_1.$$

$$2.2.2. T \leftarrow X_1, X_1 \leftarrow X_1^4 + bZ_1^4, Z_1 \leftarrow T^2Z_1^2.$$

$$3. x_3 \leftarrow X_1/Z_1.$$

$$4. y_3 \leftarrow (x + x_3) \left[(X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1Z_2) \right] (xZ_1Z_2)^{-1} + y.$$

5. Return (x_3, y_3) .

Для оценки сложности вычислений по алгоритму 9, можно получить выражение:

$$I(A_9) = (t-2)(6I_{mul} + 5I_{sqr}) + (I_{inv} + 11I_{mul} + 3I_{sqr}). \quad (9)$$

Наш анализ показал, что существует дополнительная возможность улучшить производительность этого алгоритма. Для этого предлагается, в п.3 алгоритма 8, не вычислять непосредственно инверсию Z_1 , т.к. она может быть получена как $Z_1^{-1} = x \cdot Z_2 \cdot (xZ_1Z_2)^{-1}$ в процессе вычислений п.4. Преимуществом алгоритма 8 перед алгоритмами типа 5, является отсутствие необходимости в хранении результатов предвычислений.

Алгоритм 10 представляет модификацию алгоритма 9, предложенную авторами, в которой используются проективные координаты Лопеса-Дахаба [6].

Алгоритм 10. Умножение Монтгомери в проективных координатах Лопеса-Дахаба

Вход: $k = (k_{t-1}, \dots, k_1, k_0)_2$, $k_{t-1} = 1$, $P = (x, y) \in E(GF(2^m))$

Выход: kP

1. Вычисление $(P, 2P)$: $X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2.$

2. For $i = t-2$ downto 0 do

2.1. If $k_i = 1$ then

$$2.1.1. A \leftarrow X_1Z_2, B \leftarrow X_2Z_1, E \leftarrow A+B, T \leftarrow E^2, X_1 \leftarrow xT + AB,$$

$$2.1.2. Z_1 \leftarrow T, C \leftarrow X_2^2, D \leftarrow Z_2^2, Z_2 \leftarrow CD, X_2 \leftarrow C^2 + bD^2.$$

2.2. else

$$2.2.1. A \leftarrow X_2Z_1, B \leftarrow X_1Z_2, E \leftarrow A+B, T \leftarrow E^2, X_2 \leftarrow xT + AB,$$

$$2.2.2. Z_2 \leftarrow T, C \leftarrow X_1^2, D \leftarrow Z_1^2, Z_1 \leftarrow CD, X_1 \leftarrow C^2 + bD^2.$$

$$3. x_r \leftarrow X_1/Z_1.$$

$$4. C \leftarrow Z_1Z_2.$$

$$5. y_r \leftarrow (x + x_r)(X_1X_2 + xE + yC)(xC)^{-1} + y.$$

6. Return (x_r, y_r) .

Сложность алгоритма 10 составляет:

$$I(A_{10}) = (t-2)(6I_{mul} + 5I_{sqr}) + (I_{inv} + 10I_{mul}). \quad (10)$$

Аналогично алгоритму 9, в п. 3, можно не выполнять инверсию Z_1 , т.к. она может быть получена из $x \cdot Z_2 \cdot (xC)^{-1}$.

Рассмотрим теперь подходы к уменьшению сложности СУ, основанные на использовании предвычислений. Следуя [3], воспользуемся представлением числа k в системе исчисления по основанию 2^w в виде: $(k_{d-1}, \dots, k_0)_{2^w}$, где $d = \lceil l/w \rceil$. Пусть точка Q_i такая, что $Q_i = \sum_{i:k_i=j} 2^{wi} P$. Тогда произведение kP можно представить в виде [3]:

$$\begin{aligned}
kP &= \sum_{i=0}^{d-1} k_i (2^{wi} P) = \sum_{j=1}^{2^w-1} \left(j \sum_{i:k_i=j} 2^{wi} P \right) = \sum_{j=1}^{2^w-1} jQ_j \\
&= Q_{2^{w-1}} + (Q_{2^{w-1}} + Q_{2^{w-2}}) + \dots + (Q_{2^{w-1}} + Q_{2^{w-2}} + \dots + Q_1),
\end{aligned} \tag{11}$$

Результат (11) позволил авторам работы [3] предложить алгоритм представленный ниже

Алгоритм 11. «Оконное» умножение точки

Вход: $k = (k_{t-1}, \dots, k_1, k_0)_{2^w}$, ширина окна w , $d = \lceil t/w \rceil$, $P = (x, y) \in E(GF(2^m))$

Выход: kP

1. Вычисление: $Q_i = 2^{wi} P$, $i = 0, d-1$.

2. $A \leftarrow O$, $B \leftarrow O$.

3. For $j = 2^{w-1}$ downto 1 do

3.1. For each i for which $k_i = j$ do

3.1.1. $B \leftarrow B + Q_i$.

3.2. $A \leftarrow A + B$.

4. Return A .

В предположении, что вероятности $\Pr\{(k_{(w-1)+i}, \dots, k_i) = 0\} = 1/2^w$ и $\Pr\{(k_{(w-1)+i}, \dots, k_i) \neq 0\} = (2^w - 1)/2^w$, для оценки сложности алгоритма 11 в среднем и худшем случаях, соответственно, можно получить выражения:

$$I(A_{11}^{aver}) = ((d(2^w - 1)/2^w - 1) + (2^w - 2))I_{add}, \tag{12}$$

$$I(A_{11}^{worst}) = (d + (2^w - 2))I_{add}. \tag{13}$$

Этот алгоритм можно дополнительно улучшить, если на шаге предвычислений, учесть, что количество операций удвоения соответствует количеству четных чисел, меньших 2^w , а количество операций сложения соответствует количеству нечетных чисел, меньших 2^w . Тогда сложность шага предвычислений для алгоритма 11 будет равна:

$$I(A_{11}^{pre}) = (2^{w-1} - 1)I_{add} + (2^{w-1} - 1)I_{dbl}. \tag{14}$$

Предвычисления используются и в работе [5] при построении оригинального алгоритма СУ, получившего название Lim-Lee [14]. который является модификацией алгоритма Brickell–Gordon–McCurley–Wilson (BGMW) [30]. Кратко остановимся на основной идее этого алгоритма.

Скалярный множитель k длиной t представляется в виде h блоков длины $a = \lceil t/h \rceil$, каждый такой блок обозначается через K_i . Далее каждый блок K_i представляется в виде v блоков длины $b = \lceil a/v \rceil$. Таким образом:

$$k = \sum_{r=0}^{h-1} \sum_{s=0}^{v-1} \sum_{t=0}^{b-1} k_{vbr+bs+t} 2^{vbr+bs+t}. \tag{15}$$

Авторами [5] предлагается представить в соответствии с (15) произведение kP следующим образом

$$kP = \sum_{t=0}^{b-1} 2^t \sum_{s=0}^{v-1} P_{s,u}, \quad (16)$$

где $P_{s,u}$ - массив предвычисленных значений для $s \in [0, v[$, $u \in [1, 2^h[$, $u = (u_{h-1}, \dots, u_0)_2$, причем

$$u = I_{s,t} = \sum_{r=0}^{h-1} k_{vbr+bs+t} 2^r, \quad (17)$$

где $s \in [0, v-1[$, $t \in [0, b[$.

Элементы массива предвычислений определяются следующим образом:

$$P_{0,u} = \sum_{r=0}^{h-1} u_r 2^{rvb} P, \quad P_{s,u} = 2^{sb} P_{0,u}. \quad (18)$$

Выбор параметров h и v приводит к выбору между производительностью и объемом памяти, которую можно выделить для предвычислений.

Алгоритм 12. Умножение Lim-Lee фиксированной точки

Вход: $k = (k_{t-1}, \dots, k_1, k_0)_2$, h , v , $s \in [0, v[$, $u \in [1, 2^h[$, $P = (x, y) \in E(GF(2^m))$

Выход: kP

-
1. For $u = 1$ to $2^h - 1$ do
 - 1.1. For $s = 0$ to $v - 1$ do
 - 1.1.1. $u \leftarrow (u_{h-1}, \dots, u_1, u_0)_2$.
 - 1.1.2. $P_{s,u} \leftarrow 2^{sb} \sum_{i=0}^{h-1} u_i 2^{vbi} P$.
 2. $Q \leftarrow O$.
 3. For $t = b - 1$ downto 0 do
 - 3.1. $Q \leftarrow 2Q$.
 - 3.2. For $s = v - 1$ downto 0 do
 - 3.2.1. $I_{s,t} \leftarrow \sum_{i=0}^{h-1} 2^i K_{vbi+bs+t}$.
 - 3.2.2. If $(I_{s,t} \neq 0)$ then $u \leftarrow I_{s,t}$, $Q \leftarrow Q + P_{s,u}$.
 4. Return Q .
-

Алгоритм 12 требует хранить $v(2^h - 1)$ точек. Средняя сложность и сложность в худшем случае, соответственно, составит:

$$I(A_{12}^{aver}) = (((2^h - 1)/2^h)vb - 1)I_{add} + (d - 1)I_{dbl}, \quad (19)$$

$$I(A_{12}^{worst}) = (vb - 1)I_{add} + (d - 1)I_{dbl}. \quad (20)$$

Упрощенная версия алгоритма 12 рассматривается в работах [3, 5, 12] и получила название комбинированного умножения. Согласно алгоритму, множитель k записан в виде w строк и колонки полученной матрицы обрабатываются одна за другой.

$$[a_{w-1}, \dots, a_1, a_0]P = a_{w-1} 2^{(w-1)d} P + \dots + a_1 2^d P + a_0 P, \quad (20)$$

где $d = \lceil l/w \rceil$ и $a_i \in Z_2$.

Приведем алгоритм, реализующий описанный выше подход.

Алгоритм 13. Комбинированное умножение фиксированной точки

Вход: $k = (k_{t-1}, \dots, k_1, k_0)_2$, ширина окна w , $d = \lceil t/w \rceil$, $P = (x, y) \in E(GF(2^m))$

Выход: kP .

1. Вычисление: $[(a_{w-1}, \dots, a_0)]P$, $\forall (a_{w-1}, \dots, a_0) \in Z_{2^w}$, другими словами вычислить

$$\sum_{i=0}^{2^w-1} a_i 2^{di} P, \forall (a_{w-1}, \dots, a_0) \in Z_{2^w}.$$

2. Представить множитель k в виде:

$$k = k_{d(w-1)+(d-1)} 2^{d(w-1)+(d-1)} + \dots + k_{d(w-1)} 2^{d(w-1)} + k_{d(w-2)+(d-1)} 2^{d(w-2)+(d-1)} + \dots + k_{d(w-w)} 2^{d(w-w)}.$$

3. $Q \leftarrow O$.

4. For $i = d-1$ downto 0 do

4.1. $Q \leftarrow 2Q$.

4.2. $Q \leftarrow Q + [(k_{d(w-1)+i}, \dots, k_{d(w-w)+i})]P$.

5. Return Q .

В п. 4.2. может возникнуть ситуация, когда $(k_{d(w-1)+i}, \dots, k_{d(w-w)+i}) = 0$, то операция сложения не выполняется. Следовательно $P\{(k_{d(w-1)+i}, \dots, k_{d(w-w)+i}) = 0\} = 1/2^w$ и наоборот $P\{(k_{d(w-1)+i}, \dots, k_{d(w-w)+i}) \neq 0\} = (2^w - 1)/2^w$, исходя из этого сложность в среднем и сложность в худшем случае алгоритма 14, соответственно, составит:

$$I(A_{13}^{aver}) = ((d-1)(2^w - 1)/2^w)I_{add} + (d-1)I_{dbl}, \quad (21)$$

$$I(A_{13}^{worst}) = (d-1)I_{add} + (d-1)I_{dbl}. \quad (22)$$

Сложность шага 1, предвычислений, алгоритма 13 равна:

$$I(A_{13}^{pre}) = w2^{w-1}I_{add} + d(w-1)I_{dbl}. \quad (23)$$

Количество операций сложения соответствует количеству разрядов $a_i = 1$, $\forall (a_{w-1}, \dots, a_0) \in Z_{2^w}$ в (23), другими словами $\sum_{i=1}^w iC_w^i = w2^{w-1}$. Количество операций удвоения соответствует степени множителя $2^{(w-1)d}$ в (23).

Как указано в [5], комбинированный метод с фиксированным основанием в отличается от оконного метода с фиксированным основанием необходимостью в незначительно меньшем объеме памяти.

Другой разновидностью алгоритма 13 является алгоритм de Rooij [9], представлен алгоритмом 14, но требующий значительно меньших объемов предвычислений.

Алгоритм 14. Умножение de Rooij фиксированной точки

Вход: $k = (k_{d-1}, \dots, k_1, k_0)_{2^w}$, заранее предвычислены $P_i = 2^{(i-1)w} P, i = \overline{1, d}$, $P = (x, y) \in E(GF(2^m))$

Выход: kP

1. Определить $M \in [0, d-1]$ такой, что $z_M \geq z_i$ для $i = \overline{0, d-1}$.

2. Определить $N \in [0, d-1]$, $N \neq M$ такой, что $z_N \geq z_i$ для всех $i = \overline{0, d-1}$, $i \neq M$.

3. For $i = 0$ to $d-1$ do

3.1. $z_i \leftarrow k_i$.

4. Вычислить M и N для (z_{d-1}, \dots, z_0) .

5. While $z_N \geq 0$ do
 - 5.1. $q \leftarrow \lfloor z_M/z_N \rfloor$, $P_N \leftarrow qP_M + P_N$, $z_M \leftarrow z_M \pmod{z_N}$.
 - 5.2. Вычислить M и N для (z_{d-1}, \dots, z_0) .
6. $C \leftarrow z_M P_M$.
7. Return (C) .

Заметим, что п. 5.1 требует умножения P_M на q , где $0 \leq q \leq 2^w$, которое выполняется посредством одного из описанных выше алгоритмов.

Сложность алгоритма 14 составляет $\log_2 k + W_H(k)$ групповых операций, где $W_H(k)$ – вес по Хеммингу множителя k . В среднем, сложность составляет $1.5 \log_2(k)$ групповых операций. Используя NAF представление множителя и «окно» предвычислений, сложность может быть уменьшена до $1.2 \log_2(k)$ групповых операций.

Число предвычисленных точек равно $2^w - 2$, значение w определяется из условия $2^{(d+2)w} \geq \#E(GF(p^m))$, где $d = \lceil t/w \rceil$.

Рассмотрим случай, когда умножается базовая точка P , являющаяся фиксированной, тогда СУ точки может быть ускорено за счет предвычисления значений $2P, 2^2P, \dots, 2^{t-1}P$, откуда следует, что сложность алгоритма 1 двоичного умножения справа налево составит:

$$I(A_{15}) = \lceil m/2 \rceil I_{add}. \quad (24)$$

В работе [10] предлагаются выражения для вычисления $2^i P$, $i = \overline{1, 4}$ на $E(GF(2^m))$ в аффинном представлении точек. Формулы с уменьшенной сложностью приведены в работе [11]. Их эффективность обусловлена граничным отношением $T_{inv}/T_{mul} \geq 2.3$ [11] в поле $GF(2^m)$. Для проективных координат подобный подход не эффективен [11]. Аналогичный [10, 11], но менее эффективный подход описан в [12].

В том же ракурсе следует отметить материалы [33, 34] посвященные эффективному вычислению выражений $2P+Q$, $3P+Q$, $4P+Q$ и $2P \pm Q$, $3P \pm Q$, $4P \pm Q$, $P, Q \in E(GF(2^m))$.

Альтернативой к описанным выше подходам, для повышения скорости вычисления $2^i P$, на $E(GF(2^m))$, являются результаты [3, 5], которые представлены как алгоритм 16. Условием применения алгоритма является $T_{inv}/T_{mul} < 1$ [5]. Основная идея алгоритма заключается в том, что точка $P(x, y) \in E(GF(2^m))$, $x \neq 0$ может быть представлена парой (x, λ) . Элемент λ вычисляется как $\lambda \leftarrow x + y/x$, но при этом пара (x, λ) не является точкой ЭК. Для выполнения операции удвоения $2P$, представление точки в виде $P(x, \lambda)$, по сравнению с аффинным представлением $P(x, y)$, позволяет сэкономить одну операцию умножения.

Алгоритм 16. Удвоение в аффинных координатах (эффективное инвертирование в поле)

Вход: k , $P = (x, y) \in E(GF(2^m))$

Выход: $2^k P$

1. $\lambda \leftarrow x + y/x$.
2. For $j = 1$ to $k - 1$ do
 - 2.1. $x_2 \leftarrow \lambda^2 + \lambda + a$, $\lambda_2 \leftarrow \lambda^2 + a + b/(x^4 + b)$, $x \leftarrow \lambda^2$, $\lambda \leftarrow \lambda_2$.
3. $x_2 \leftarrow \lambda^2 + \lambda + a$, $y_2 \leftarrow x^2 + (\lambda + 1)x_2$.
4. Return (x_2, y_2) .

Сложность алгоритма 16 составит:

$$I(A_{16}) = (k-1)I_{inv} + kI_{mul} + (3(k-2)+2)I_{sqr}. \quad (24)$$

Предложенная авторами модификация предыдущего алгоритма представлена алгоритмом 17.

Алгоритм 17. Удвоение в стандартных проективных координатах

Вход: $k, P = (x, y) \in E(GF(2^m))$

Выход: $2^k P$

-
1. $X \leftarrow x, Y \leftarrow y, Z \leftarrow 1.$
 2. $\lambda \leftarrow X^2 + Y, Z_\lambda \leftarrow X.$
 2. For $j = 1$ to $k-1$ do
 - 2.1. $L \leftarrow \lambda^2, Z_2 \leftarrow Z_\lambda^2, A \leftarrow a \cdot Z_2, X_2 \leftarrow L + \lambda \cdot Z_\lambda + A.$
 - 2.2. $Z_{\lambda_2} \leftarrow Z_2 \cdot C, B \leftarrow b \cdot Z^4, C \leftarrow (X^4 + B), \lambda_2 \leftarrow (L + Z_2) \cdot C + B \cdot Z_2.$
 - 2.3. $X \leftarrow L, Z \leftarrow Z_2, \lambda \leftarrow \lambda_2, Z_\lambda \leftarrow Z_{\lambda_2}.$
 3. $\lambda \leftarrow \lambda/Z_\lambda, x_2 \leftarrow \lambda^2 + \lambda + a, x_2 \leftarrow X_2/Z_2, y_2 \leftarrow x^2 + (\lambda+1)x_2.$
 4. Return (x_2, y_2)
-

В приведенном алгоритме операции над точками выполняются в стандартных проективных координатах, что позволяет исключить инвертирование в поле. Этот алгоритм следует позиционировать, как алгоритм с эффективным умножением в поле, в отличие от алгоритма 16.

Сложность алгоритма 17 составляет:

$$I(A_{17}) = 2I_{inv} + (4(k-2)+3)I_{mul} + (6(k-2)+2)I_{sqr}. \quad (25)$$

Следующим способом позволяющим снижать сложность СУ является использование смешанных проективных координат [6], это позволит сэкономить одну операцию умножения на каждом сложении точек.

В различных криптографических схемах, таких как ECDSA, ECGDSA, ECKDSA, при проверке цифровой подписи возникает необходимость в вычислении выражения $kP + lQ$. В произведении kP , точка P - фиксирована, а в lQ , Q - произвольная. Авторы работ [5, 20] предлагают одновременно умножать и вычислять сумму. Алгоритм 18, реализует этот подход.

Алгоритм 18. Одновременное умножение

Вход: $k = (k_{d-1}, \dots, k_1, k_0)_{2^w}, l = (l_{d-1}, \dots, l_1, l_0)_{2^w}, d = \lceil t/w \rceil, P, Q \in E(GF(2^m))$

Выход: $kP + lQ$

-
1. Вычисление: $D_{i,j} \leftarrow iP + jQ, i = 0, 2^w - 1, j = 0, 2^w - 1.$
 2. $R \leftarrow O.$
 3. For $i = d-1$ downto 0 do
 - 3.1. $R \leftarrow 2^w \cdot R.$
 - 3.2. $R \leftarrow R + D_{i,j}.$
 4. Return $R.$
-

Алгоритм хранит предвычисленными 2^{2w} точек ЭК.

Сложность алгоритма 18 составляет:

$$I(A_{18}) = (2^{2w} - 3)I_{add} + (d-1)(2^{2w} - 1)/2^{2w} I_{add} + (d-1)wI_{dbl}. \quad (26)$$

Сложность этапа предвычислений алгоритма 18 равна:

$$I(A_{18}^{pre}) = 2((2^{w-1} - 1)I_{add} + (2^{w-1} - 1)I_{dbl}) + (2^{2w} - 2^{w+1} + 1)I_{add}. \quad (27)$$

Сложность этапа предвычислений возможно уменьшить. Если все необходимые предвычисления для базовой точки сделать во время инициализации системы, то первое слагаемое уменьшится в 2 раза.

Авторами предложено развитие подхода лежащего в основе алгоритма 18 посредством идеи алгоритмов 10 и 12. Получившийся алгоритм был назван как алгоритм 19 - одновременного умножения на основе умножения Lim-Lee фиксированной точки, алгоритм 12, и умножения Монтгомери в проективных координатах Лопеса-Дахаба, алгоритм 10. Каждый из них выполняется последовательно один за другим, в конце, результаты складываются.

Сложность алгоритма 19, соответственно, составит:

$$I(A_{19}^{aver}) = (d(2^w - 1)/2^w)I_{add} + dI_{dbl} + (t-2)(6I_{mul} + 5I_{sqr}) + (1I_{inv} + 11I_{mul} + 3I_{sqr}), \quad (28)$$

Схожие относительно идеи алгоритма, были рассмотрены в работе [3]. Отличие состоит в том, что в качестве алгоритма умножения базовой точки был использован алгоритм 13, который обладает большей вычислительной сложностью в сравнении с алгоритмом 12.

Ниже приведен алгоритм одновременного умножения предложенный в [23].

Алгоритм 20. Одновременное умножение Shamir'a

Вход: $k = (k_{t-1}, \dots, k_1, k_0)_{2^w}$, $l = (l_{t-1}, \dots, l_1, l_0)_{2^w}$, $P, Q \in E(GF(2^m))$

Выход: $kP + lQ$

-
1. Вычисление: $G = P + Q$.
 2. $A \leftarrow O$.
 3. For $i = t-1$ downto 0 do
 - 3.1. $A \leftarrow 2A$.
 - 3.2. If $(k_i, l_i) = (1, 0)$ then $A \leftarrow A + P$
 - 3.3. else If $(k_i, l_i) = (0, 1)$ then $A \leftarrow A + Q$
 - 3.3.1. else If $(k_i, l_i) = (1, 1)$ then $A \leftarrow A + G$
 4. Return (A).
-

Сложность алгоритма 20 и этапа предвычислений, соответственно, составит:

$$I(A_{20}^{aver}) = (\frac{3}{4}t + 1)I_{add} + (t-1)I_{dbl}, \quad (30)$$

$$I(A_{20}^{pre}) = 1I_{add}. \quad (31)$$

Известно, что число сложений точек зависит от общего веса по Хеммингу чисел k и l длиной t . Им ставится в соответствие матрица размером $2 \times t$. В ней строки соответствуют двоичным знаковым или беззнаковым представлением этих чисел. *Общий вес по Хеммингу* $JHW(k, l)$ (JHW – joint Hemming weight) чисел k и l - число отличных от нуля строк этой

матрицы. Например, если k и l имеют веса по Хеммингу $t/2$, то $JHW(k, l) = (3/4)t$. Для NAF представления с $JHW(k, l) = \frac{5}{9}t$, сложность алгоритма 20 составит:

$$I(A_{20}^{aver} NAF) = (\frac{5}{9}t + 2)I_{add} + tI_{dbl}. \quad (32)$$

Отметим, что п.1 алгоритма 20 требует вычисления не только $P + Q$, но и $P - Q$.

Альтернативой к NAF представлению скалярного множителя считается смежное разряженное представление JSF (joint sparse form) [24]. JSF – знаковое представление, где любые три последовательные колонки матрицы включают нулевую колонку, и если два смежные строки отличны от нуля, тогда они или (11) или $(-1-1)$ и соответствующие записи из другой строки будут (± 10) . Например, два числа $(4773)_{10}$ и $(4395)_{10}$ могут быть представлены в JSF как:

$$\begin{bmatrix} 4773 \\ 4395 \end{bmatrix} = \begin{bmatrix} 1010\tilde{1}\tilde{1}0100101 \\ 1000100110\tilde{1}0\tilde{1} \end{bmatrix}.$$

В отличие от других представлений, для пары чисел k и l $JHW(k, l) = \frac{1}{2}t$. Благодаря этому замечательному свойству, сложность алгоритма 20 со скалярными множителями, представленными в JSF, составит:

$$I(A_{20}^{aver} JSF) = (\frac{1}{2}t + 2)I_{add} + tI_{dbl}. \quad (33)$$

Кардинально отличающийся подход описан в работе [16]. Суть его состоит в переходе от операции удвоения к операции деления пополам, т.е. вычисляется такая точка Q , что $2Q = P$. Этот подход применим лишь для половины $E(GF(2^m))$ с $Tr(a) = 1$. Основными операциями в этом подходе являются: извлечение квадратного корня, вычисление следа элемента, решение в поле квадратных уравнений вида $x^2 + x = s$, $s \in GF(2^m)$. Описанный выше метод не эффективен при полиномиальном представлении элементов поля, т.к. требует для достижения приемлемого быстродействия использования значительных предвычислений и большого количества памяти. В тоже время, он эффективно реализуются в оптимальном нормальном базисе [7, 17, 21, 22], при аппаратной реализации криптопреобразований.

2. Сравнение алгоритмов СУ по вычислительной сложности и выводы

С учетом сложности реализаций, представленные алгоритмы СУ можно разделить на следующие группы:

- умножение без предвычислений;
- умножение с предвычислениями;
- умножение фиксированной точки;
- одновременное умножение;
- умножение на скаляр вида 2^m , $m > 2$;
- умножение на скаляр вида 2^m , $2 < m \leq 5$.

В таблице 1 представлены результаты обобщения оценок вычислительной сложности алгоритмов скалярного умножения без предвычислений.

Таблица 1

Наименование алгоритма	Сложность
Алгоритм 1. Двоичное умножение слева направо	$I(A_1^{aver}) = \frac{t}{2}I_{add} + tI_{dbl}$
Алгоритм 3. Двоичное NAF умножение	$I(A_3) = \frac{t}{3}I_{add} + tI_{dbl}$
Алгоритм 6. Умножение в проективных	$I(A_6) = (t - 2)(\frac{1}{4}(I_{add} + I_{sub}) + I_{dbl}) + (2I_{inv} + 3I_{mul})$

координатах Якоби со сложением и вычитанием	
Алгоритм 7. Двоичное умножение Монтгомери	$I(A_7) = (t-1)I_{add} + (t-1)I_{dbl}$
Алгоритм 8. Умножение точек Монтгомери в аффинных координатах	$I(A_8) = (t-2)(2I_{mul} + 2I_{inv} + 2I_{sqr}) + (1I_{inv} + 1I_{mul} + 1I_{sqr})$
Алгоритм 9. Умножение Монтгомери в стандартных проективных координатах	$I(A_9) = (t-2)(6I_{mul} + 5I_{sqr}) + (1I_{inv} + 11I_{mul} + 3I_{sqr})$
Алгоритм 10. Умножение Монтгомери в проективных координатах Лопеса-Дахаба	$I(A_{10}) = (t-2)(6I_{mul} + 5I_{sqr}) + (1I_{inv} + 10I_{mul})$
Алгоритм 16. Удвоение в аффинных координатах (эффективное инвертирование)	$I(A_{16}) = (k-1)I_{inv} + kI_{mul} + (3(k-2) + 2)I_{sqr}$
Алгоритм 17. Удвоение в стандартных проективных	$I(A_{17}) = 2I_{inv} + (4(k-2) + 3)I_{mul} + (6(k-2) + 2)I_{sqr}$

Из данных приведенных в таблице 1, следует, что среди алгоритмов СУ без предвычислений минимальная сложность обеспечивается при использовании алгоритма 9 или 10 с авторскими улучшениями. Одним из наиболее предпочтительных, среди известных алгоритмов удвоения, без предвычислений, является алгоритм 17.

В таблице 2 представлены результаты обобщения оценки вычислительной сложности алгоритмов СУ с предвычислениями.

Таблица 2

Наименование алгоритма	Сложность
Алгоритм 5. «Оконное» умножение	$I(A_5) = (1I_{dbl} + (2^{w-2} - 1)I_{add}) + (t/(w+1)I_{add} + mI_{dbl})$
Алгоритм 11. «Оконное» умножение фиксированной точки	$I(A_{12}^{aver}) = ((d(2^w - 1)/2^w - 1) + (2^w - 2))I_{add}$
Алгоритм 12. Умножение Lim-Lee фиксированной точки	$I(A_{13}^{aver}) = (((2^h - 1)/2^h)vb - 1)I_{add} + (d - 1)I_{dbl}$
Алгоритм 13. Комбинированное умножение фиксированной точки	$I(A_{14}^{aver}) = ((d - 1)(2^w - 1)/2^w)I_{add} + (d - 1)I_{dbl}$
Алгоритм 15. Двоичное умножение слева направо фиксированной точки, с предвычисленными удвоениями	$I(A_{15}) = \lceil \frac{t}{2} \rceil I_{add}$

Анализ соотношений, приведенных в таблице 2, показывает, что среди алгоритмов умножения с предвычислениями наименьшую сложность имеет алгоритм 13, который обладает предпочтительным отношением сложность-память. При отсутствии ограничения на количество точек, которые необходимо хранить в памяти, следует выделить алгоритм 12. При аппаратной реализации наиболее предпочтительным является алгоритм 15, но при этом он требует для хранения данных предвычислений наибольшего количества памяти.

В таблице 3 представлены результаты оценки сложности одновременного скалярного умножения.

Таблица 3

Наименование алгоритма	Сложность
Алгоритм 18. Одновременное умножение	$I(A_{18}) = (2^{2w} - 3)I_{add} + (d - 1)(2^{2w} - 1)/2^{2w} I_{add} + (d - 1)wI_{dbl}$
Алгоритм 19. Одновременное умножение на основе комбинированного умножения фиксированной точки	$I(A_{19}^{aver}) = (d(2^w - 1)/2^w)I_{add} + dI_{dbl} + (t - 2)(6I_{mul} + 5I_{sqr}) + (1I_{inv} + 11I_{mul} + 3I_{sqr})$
Алгоритм 20. Одновременное умножение Shamir'a (двоичное представление).	$I(A_{20}^{aver}) = (\frac{3}{4}t + 1)I_{add} + (t - 1)I_{dbl}$
Алгоритм 20. Одновременное умножение Shamir'a (NAF)	$I(A_{20}^{aver} NAF) = (\frac{5}{9}t + 2)I_{add} + tI_{dbl}$

Алгоритм 20. Одновременное умножение Shamir'a (NAF)

$$I(A_{20}^{aver} JSF) = \left(\frac{1}{2}t + 2\right)I_{add} + tI_{dbl}$$

Анализ соотношений, представленных в таблице 3 позволяет сделать вывод о том, что среди алгоритмов одновременного умножения, использующих предвычисления, занимает авторский алгоритм 19. Среди алгоритмов не использующих предвычисления, необходимо выделить алгоритм 20, использующий JSF представление скалярного множителя.

В целом необходимо также отметить [19, 25], что алгоритмы 7-10, более предпочтительны в случаях, когда необходимо противостоять атакам на реализацию. Детальное описание этих алгоритмов представлено в работах [18, 26]. Указанные подходы повышения устойчивости к атакам на реализацию излагаются в работах [19, 25, 27].

Таким образом, представленные в статье результаты позволяют оценить сложность выполнения скалярного умножения в группе точек эллиптической кривой, выбрать из них наиболее предпочтительные в зависимости от требований и ограничений, которые накладываются на средства их реализации.

Список литературы. 1. ДСТУ 4541-2002. Інформаційні технології. Криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих. Формування та перевірка. 2. N. Koblitz. CM-curves with good cryptographic properties. Advances in cryptology. CRYPTO'91, LNCS 576, 1992, 279-287. 3. D. Hankerson, J. Lopez, A. Menezes. Software implementation of elliptic curve cryptography over binary fields. In Cetin K. Koc and C. Paar editors, Workshop and embedded systems (CHES'99) LNCS 1717, pp. 1-24, Springer-Verlag, august 2000. 4. В.Ю. Ковтун, С.А. Головашич, Ю.В. Стасев. Сравнительный анализ алгоритмов умножения и приведения по модулю в поле $GF(2^m)$. // Радиотехника: Всеукр. межвед. науч.-техн. сб. 2003. Вып. 135. С. 129-141. 5. J. Lopez, R. Dahab. An overview of elliptic curve cryptography. 2000. 6. В.Ю. Ковтун, Ю.В. Стасев, О.А. Смирнов, Я.Ю. Стасева. Анализ методов представления точек эллиптической кривой над двоичными полями. // Сборник научных трудов ХВУ. – Харьков: ХВУ. Системы обработки информации. – 2002. – Вып. 5(21). 7. IEEE P1363 / D9 (Draft Version 9). Standard Specifications for Public Key Cryptography. 8. AMERICAN NATIONAL STANDARD X9.62-1998 (Draft version), Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 9. A.D. Woodbury. Efficient algorithms for elliptic curve cryptosystems on embedded systems. Worcester Polytechnic Institute. -September, 2001. 10. J. Guajardo, C. Paar. Efficient algorithms for elliptic curve cryptosystems. Advanced in cryptology, Proc. CRYPTO' 97, LNCS 1294, pp.342-356, Springer-Verlag, 1997. 11. В.Ю. Ковтун, В.В. Вишенько, В.Я. Певнев, А.А. Смирнов. Модифікований алгоритм скалярного добутку точок еліптичної кривої над двійковими полями. Вісник Житомирського державного технологічного університету вип. 2(26) т.1, Житомир: – 2003 – С. 187-192. 12. J. Lopez, R. Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation, Proceedings of the first workshop on cryptographic hardware and embedded systems - CHES'99, LNCS 1717, pp316-327, Springer-Verlag, 1999. 13. R. Schroepel. Faster elliptic calculations in $GF(2^m)$, Preprint. 1998. 14. C.H. Lim, P.J. Lee. More flexible exponentiation with precomputation, In Advances in Cryptology-CRYPTO'94, pp95-107, Springer-Verlag, 1994. 15. K. Itoh, M. Takenaka, N. Torii, S. Temma, Y. Kurihara. Fast implementation of public-key cryptography on DSP TMS320C6201. In proceedings of the first workshop on cryptography hardware and embedded systems (CHES'99), LNCS 1717, pp61-72, Springer-Verlag, 1999. 16. E. W. Knudsen. Elliptic scalar multiplication using point halving. In Asiacrypt'99, LNCS 1716, pp.135-149, Springer-Verlag, 1999. 17. J. Solinas. An improved algorithm for arithmetic on family of elliptic curves. Advanced in cryptology, CRYPTO' 97, LNCS 1294, B. Kaliski, pp.357-371, Springer-Verlag, 1997. 18. P.C. Kocher. Timing attacks on implementations of Diffie-Helman, RSA, DSS, and other systems. Advances in cryptology. CRYPTO'96, LNCS 1109, pp.104-113, Springer-Verlag, 1996. 19. T. Izu, T. Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. Research Report CORR 02-03, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, 2002. 21 pages. 20. Y. Sakai, K. Sakurai. An efficient representation of scalars for simultaneous elliptic scalar multiplication. Special section on discrete mathematics and its applications. IEICE TRANS. FUNDAMENTALS, vol.E86-A, No.5 may 2003. 21. D.V. Bailey, C. Paar, G. Sarcozy, M. Hofri. Computation on optimal extension fields. May 2000. 22. D.V. Bailey. Optimal extension fields for fast arithmetic in public-key algorithms. April 1998. 23. T. ELGamal. Public key cryptosystem and a signature scheme based on discrete logarithms, IEEE trans. Inf. Theory, vol.31, No.4, pp.469-472, 1985. 24. J. A. Solinas. Some computational speedups and bandwidth improvements for curves over prime fields. 25. B. Moller. Securing elliptic curve point multiplication against side-channel attacks. ICS 2001. 26. K. Okeya, K. Sakurai. Power analysis breaks elliptic curve cryptosystems even secure against the timing attack. Progress in cryptology – INDOCRYPT'2000 (2000), B.K. Roy, E. Okamoto, Eds., LNCS 1997, pp.178-190. 27. W. Fischer, C. Giraud, E.W. Knudsen, J.-P. Seifert. Parallel scalar multiplication on general elliptic curves over F headed against non-differential side-channel attacks. 28. Chi-Sung, Wen-Chung Kuo. Speeding up the computations of elliptic curves crypto schemes. Computers math. Applic., vol. 33, No. 5, pp. 29-36, 1997. 29. A. Weimerskirch, C. Paar, S. C. Shantz. Elliptic curve cryptography on a Palm OS device. The 6th Australasian conference on information security and privacy, ACISP'01, 2001. 30. E.F. Brickell, D.M. Gordon, K.S. McCurley, D.B. Wilson. Fast exponentiation with

precomputation (Extended abstract). 31. *J.A. Mur, D.R. Stinson*. Minimality and other properties of the width- w nonadjacent form. Research Report CORR 04-08, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, 2004. 32. *J.A. Mur, D.R. Stinson*. Alternative digit sets for nonadjacent representation. Research Report CORR 03-04, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, 2003. 33. *M. Ciet, M. Joye, K. Lauter*. Trading inversions for multiplications in elliptic curve cryptography. Available at: www.dice.ucl.ac.be/crypto/. 34 *K. Eisentrager, K. Lauter, P. L. Montgomery*. An efficient procedure to double and add points on an elliptic curve. August 5, 2002. 35. FIPS 186-2. Recommended elliptic curves for federal government use. July 1999. 36. ISO/IEC FCD 15946-2: Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 2: Digital signatures, Final Committee Draft, 1999. 37. ISO/IEC FCD 15946-4: Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 4: Digital signatures giving message recovery, Final Committee Draft, 2001.