Comparison of Algorithms for Elliptic Curve Cryptography over Finite Fields of $GF(2^m)$

The IASTED International Conference on
Communication, Network, and Information Security
CNIS 2003, December 10-12, 2003
New York, USA

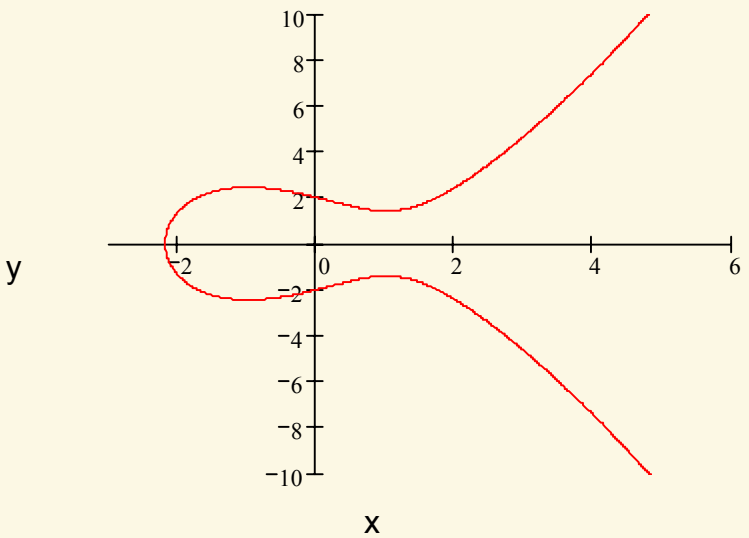
Mathias Schmalisch · Dirk Timmermann

Contents

Motivation

- Elliptic Curve Cryptography (ECC) uses smaller key length than other public key algorithms at the same level of security
  - ECC uses today 160 bit
  - RSA uses today 1024 bit
- ECC can be used for authentication, encryption, digital signatures, key exchange and so forth
- Because of the short key length ECC is interesting for small and embedded devices
- For ECC the scalar multiplication $k*P$ must be computed
  - There exist several algorithms for the scalar multiplication
  - What algorithm is the best for a software or hardware computation?

Elliptic Curves



- Weierstrass equation:

  $E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$

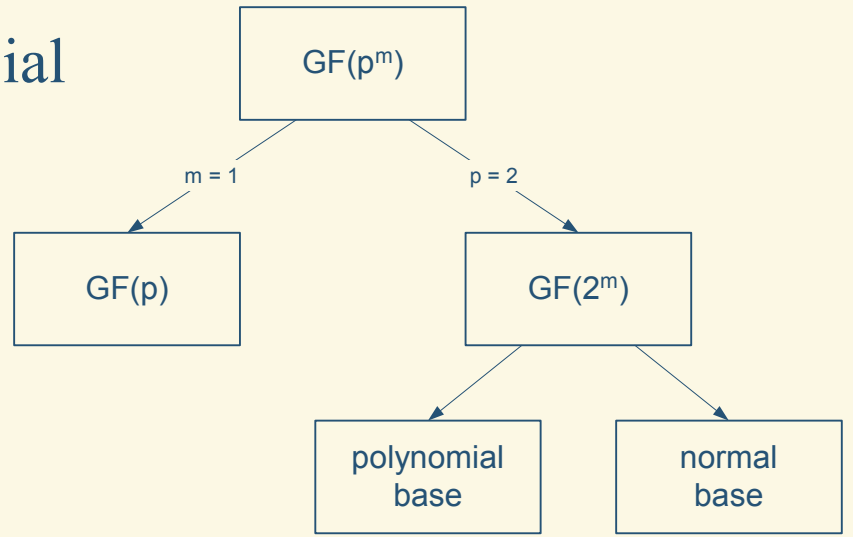
  $x, y, a_i \in GF(q)$

- E is the set of solutions of the Weierstrass equation in the affine plane
- Elliptic curves used in cryptography will be mapped on finite fields $GF(q)$
- The curve may not be singular
  - Discriminant $\Delta \neq 0$

Finite Fields $GF(q)$

- For finite fields there exist two possible notations
  - $GF(q) = \boldsymbol{F}_q$
- Where $q = p^m$ with p a prime number and m a nonnegative integer
- For cryptography there only the special cases $m = 1$ or $p = 2$ are from interest
- For hardware computation the special case $p = 2$ is very interesting
  - char($GF(2^m)$) = 2

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



Point Operations

- Point addition $P \neq Q$
- A line through P and Q

$$\lambda = (x_1 + x_2)/(y_1 + y_2)$$

- Point doubling $P = Q$
- A tangent at P

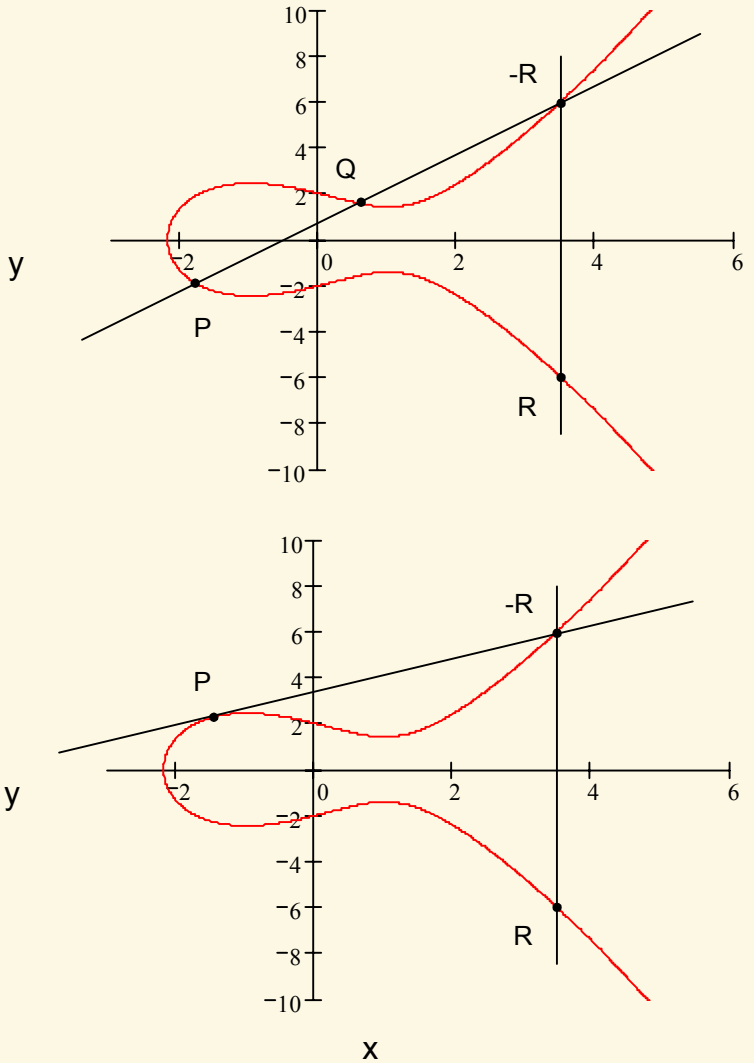
$$\lambda = x_1 + y_1/x_1$$

- Computing the coordinates of R

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \qquad \text{if } P \neq Q$$

$$y_3 = \lambda x_3 + x_3 + x_1{}^2 \qquad \text{if } P = Q$$



Affine vs. Projective Coordinates

Affine Coordinates (x, y)

Advantage:

- Less finite field operations
- Easy equations for point operations

Disadvantage:

- One inversion per point operation

Projective Coordinates (X, Y, Z)

Advantage:

- No inversion for point operations

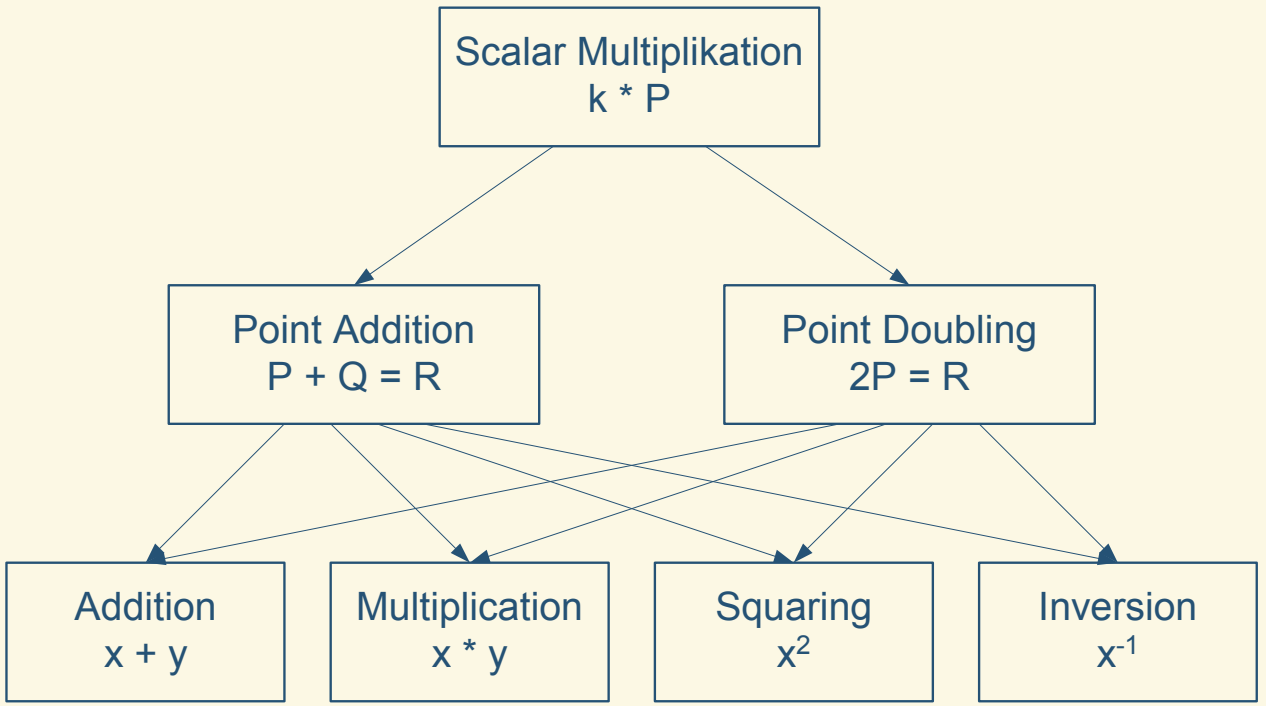
Disadvantage:

- Many finite field operations
- Conversion/Back conversion from/to affine coordinates

	Add	Mult	Sqr	Inv
Adding	8	2	1	1
Doubling	5	2	2	1

	Add	Mult	Sqr
Adding	7	15	5
Doubling	4	5	5

Scalar Multiplication *k*P*



Scalar Multiplikation

Point Operations on Ellitpc Curve

Finite Field Operations

Algorithms for Scalar Multiplication *k*P*

Double and Add (DaA)

Input: An integer $k > 0$ and a point P

Output: $Q = k * P$

1. $k = (k_{n-1}, ..., k_1, k_0)_2$
2. $Q = P$
3. for i in $(n - 2)$ downto 0 do
4. $\quad Q = 2 * Q$
5. $\quad$ if k_i = "1" then
6. $\quad\quad Q = Q + P$
7. return Q

- Simplest algorithm
- Needs $(n - 1)$ point doublings
- The number of point aditions depends on the Hamming weight $Hw(k)$
  - $(Hw(k) - 1)$ point additions
  - on average $Hw(k) = n / 2$

Algorithms for Scalar Multiplication *k*P*

- Reducing the Hamming weight

Double and Add/Subtract (DaAS)

- Signed Digit Representation
  - Elements are 0, 1, –1
  - on average $Hw(k) = n / 3$
- Conversion algorithms
  - Canonical Recoding Algorithm
  - Modified Booth
- Additionally the point subtraction is needed
  - $-P = (x, y + x)$

Input: An integer $k > 0$ and a point P
Output: $Q = k * P$

1. $k = (k_{n-1}, ..., k_1, k_0)_{SD}$ $k_i \in \{0,1,-1\}$
2. $Q = P$
3. for i in $(n - 2)$ downto 0 do
4. $Q = 2 * Q$
5. if k_i = "1" then
6. $Q = Q + P$
7. else if k_i = " –1" then
8. $Q = Q – P$
9. return Q

Improvement for DaAS Algorithm

- If a point addition is faster to compute than a point doubling, then a speed up is possible
  - $Q_i = 2Q_{i\text{-}1} + P = (Q_{i\text{-}1} + P) + Q_{i\text{-}1}$
- Method of K. Eisenträger, K. Lauter and P. L. Montgomery
  - At the computation of successive point additions the computation of the y-coordinate can be saved

$$\lambda_3 = \frac{y_2 + y_1}{x_2 + x_1} \qquad \lambda_4 = \frac{y_3 + y_2}{x_3 + x_2}$$

$$x_3 = \lambda_3^{\,2} + \lambda_3 + x_1 + x_2 + a \qquad x_4 = \lambda_4^{\,2} + \lambda_4 + x_3 + x_2 + a$$

$$y_3 = \lambda_3 \left(x_2 + x_3\right) + x_3 + y_2 \qquad y_4 = \lambda_4 \left(x_3 + x_4\right) + x_4 + y_3$$

$$\lambda_4 = \frac{y_3 + y_2}{x_3 + x_2} = \frac{\lambda_3 \left(x_2 + x_3\right) + x_3 + y_2 + y_2}{x_3 + x_2} = \lambda_3 + \frac{x_3}{x_3 + x_2}$$

Montgomery Algorithm

Input: An integer $k > 0$ and a point P

Output: $Q = k * P$

1. $k = (k_{n-1}, ..., k_1, k_0)_2$
2. $P_1 = P; P_2 = 2P$
3. for i in $(n - 2)$ downto 0 do
4. if k_i = "1" then
5. $P_1 = P_1 + P_2; P_2 = 2P_2$
6. else
7. $P_1 = 2P_1; P_2 = P_1 + P_2$
8. return $(Q = P_1)$

- Publication of P. L. Montgomery
- Independent of the Hamming weight $Hw(k)$
- Computing of the y-coordinates can be saved
- y-coordinates of the sum of two points whose difference is known can be computed in terms of the x-coordinates of the involved points

Comparing the Algorithms (Software)

- Algorithms tested with two software library's for ellitpic curve cryptography
  - LiDIA
  - Cryptix Elliptix
- Measured the time of the finite field operations
  - 100000 iterations per finite field operations
- Example:
  - LiDIA
  - GF(2^{163})

Field Width *m*	Add.	Mult.	Sqr.	Inv.
163	10	271	45	2944

Coordinates	Algorithm	Time (ms)
Affine	DaAS	7.35
	DaAS Imp.	7.60
	Montgomery	10.73
Projective	DaAS Men.	3.84
	DaAS IEEE	3.48
	DaAS LD	3.33
	Montgomery	2.74

Comparing the Algorithms (Hardware)

- Addition and squaring can be computed in one clock cycle
  - Only XOR combinations
- A serial multiplications need m clock cycles
- For inversion there exist a fast solution from H. Brunner, A. Curiger and M. Hofstetter
  - Serial implementation
  - Needs only m clock cycles
- For projective coordinates Fermat's theorem can be used

Field Width m	Add.	Mult.	Sqr.	Inv.
163	1	m	1	m

Coordinates	Algorithm	Clock Cycles
Affine	DaAS	108266
	DaAS Imp.	99246
	Montgomery	108077
Projective	DaAS Men.	306419
	DaAS IEEE	276158
	DaAS LD	195636
	Montgomery	163987

Conclusion

- Overview about known algorithms for scalar multiplication
- The time for the finite field inversion is deciding, what coordinate system is the best
- Comparing the algorithms for software and hardware computation
  - Software: Montgomery Algorithm in projective coordinates
  - Hardware: DaAS Algorithm and Improvement in affine coordinates
- Fastest hardware solution uses other algorithm, so a speed up of up to 40% with DaAS + Improvement is possible

Thank you!