Modular Multiplication in $GF(p^k)$ using Lagrange Representation

Jean-Claude Bajard, Laurent Imbert, and Christophe Nègre

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
LIRMM – CNRS UMR 5506
161 rue Ada, 34392 Montpellier Cedex 5, France
{bajard,imbert,negre}@lirmm.fr

Abstract. In this paper we present a new hardware modular multiplication algorithm over the finite extension fields $GF(p^k)$ where $p > 2k$. We use an alternate polynomial representation of the field elements and a Lagrange like interpolation technique. We describe our algorithm in terms of matrix operations and point out some properties of the matrices that can be used to improve the hardware design. The proposed algorithm is highly parallelizable and seems well suited for hardware implementation of elliptic curve cryptosystems.

Keywords: Finite fields, multiplication, cryptography, fast implementation.

1 Introduction

Cryptographic applications such as elliptic or hyperelliptic curves cryptosystems (ECC, HECC) [1–3] and the Diffie-Hellman key exchange algorithm [4] require arithmetic operations to be performed in finite fields. An efficient arithmetic in these fields is then a major issue for lots of modern cryptographic applications [5]. Many studies have been proposed for the finite field $GF(p)$, where p is a prime number [6] or the Galois field $GF(2^k)$ [7–9]. In 2001, D. Bailey and C. Paar proposed an efficient arithmetic in $GF(p^k)$ when p is a pseudo-Mersenne prime [10], but although it could result in a wider choice of cryptosystems, arithmetic over the more general finite extension fields $GF(p^k)$, with $p > 2$, has not been extensively investigated yet. Moreover it has been proved that elliptic curves defined over $GF(p^k)$ – where the curves verify the usual conditions of security – provide at least the same level of security than the curves usually defined over $GF(2^k)$ or $GF(p)$.

This paper introduces a Montgomery like modular multiplication algorithm in $GF(p^k)$ for $p > 2k$. Given the polynomials $A(X)$ and $B(X)$ of degree less than k, and $G(X)$ of degree k (we will give more details on $G(X)$ in section 3), our algorithm computes $A(X) \times B(X) \times G(X)^{-1} \mod N(X)$, where both the operands and the result aregiven in an alternate representation.

In the classical polynomial representation, we consider the field elements of $GF(p^k)$ as polynomials of degree less than k in $GF(p)[X]$ and we represent the field with respect to an irreducible polynomial $N(X)$ of degree k over $GF(p)$. Any element A of $GF(p^k)$ is then represented using a polynomial $A(X)$ of degree $k-1$ or less with coefficients in $GF(p)$, i.e., $A(X) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$, where $a_i \in \{0, \ldots, p-1\}$. In this paper we consider an alternate solution which consists of representing the polynomials with their values at k distinct points instead of their k coefficients. As a result, if we choose k points $(e_1, e_2, \ldots, e_k)$, we represent the polynomial $A(X)$ with the sequence $(A(e_1), A(e_2), \ldots, A(e_k))$. Within this representation addition, subtraction and multiplication are performed over completely independent channels which is a great advantage from a chip design viewpoint.

2 Montgomery Multiplication in $GF(p^k)$

In 1985, Peter Montgomery proposed an integer reduction algorithm that can be easily extended to modular multiplication of large integers [11]. This method has recently been adapted to modular multiplication in $GF(2^k)$ (defined according to the k-order polynomial X^k) [7] and extend easily to $GF(p^k)$, with $p > 2$.

As in [7], we represent the field $GF(p^k)$ with respect to the monic irreducible polynomial X^k and we consider the field elements as polynomials of degree less than k in $GF(p)[X]$. If $A(X)$ and $B(X)$ are two elements of $GF(p^k)$, Montgomery's multiplication technique is used to compute $A(X) \times B(X) \times X^{-k} \bmod N(X)$. We successively evaluate:

$$\begin{aligned} Q(X) &= \quad -A(X)\,B(X)\,N(X)^{-1} \bmod X^k \\ R(X) &= [A(X)\,B(X) + Q(X)\,N(X)] \times X^{-k} \end{aligned}$$

2.1 Implementation

Let us denote

$$\begin{aligned} A(X) &= a_0 + a_1 X + a_2 X^2 + ... + a_{k-1} X^{k-1} \\ B(X) &= b_0 + b_1 X + b_2 X^2 + ... + b_{k-1} X^{k-1} \\ N(X) &= n_0 + n_1 X + n_2 X^2 + ... + n_{k-1} X^{k-1} \\ N^{-1}(X) &= n'_0 + n'_1 X + n'_2 X^2 + ... + n'_{k-1} X^{k-1} \end{aligned}$$

The reduction modulo X^k can be accomplished by ignoring the terms of order larger than or equal to k. Division by X^k simply consists of shifting the polynomial to the right by k places. These operations are easily integrated in the matrix operations and the computations are then decomposed as follow:

$$Q(X) = -\begin{pmatrix} n'_0 & 0 & ... & 0 & 0 \\ n'_1 & n'_0 & ... & 0 & 0 \\ \vdots & & & & \\ n'_{k-2} & n'_{k-3} & ... & n'_0 & 0 \\ n'_{k-1} & n'_{k-2} & ... & n'_1 & n'_0 \end{pmatrix} \begin{pmatrix} a_0 & 0 & ... & 0 & 0 \\ a_1 & a_0 & ... & 0 & 0 \\ \vdots & & & & \\ a_{k-2} & a_{k-3} & ... & a_0 & 0 \\ a_{k-1} & a_{k-2} & ... & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{k-2} \\ b_{k-1} \end{pmatrix}.$$

The evaluation of $R(x)$ then rewrites $R(X) =$

$$\begin{pmatrix} 0 & a_{k-1} & \dots & a_2 & a_1 \\ 0 & 0 & \dots & a_3 & a_2 \\ \vdots & & & & \\ 0 & 0 & \dots & a_{k-1} & a_{k-2} \\ 0 & 0 & \dots & 0 & a_{k-1} \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{k-3} \\ b_{k-2} \\ b_{k-1} \end{pmatrix} + \begin{pmatrix} 0 & n_{k-1} & \dots & n_2 & n_1 \\ 0 & 0 & \dots & n_3 & n_2 \\ \vdots & & & & \\ 0 & 0 & \dots & n_{k-1} & n_{k-2} \\ 0 & 0 & \dots & 0 & n_{k-1} \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{k-3} \\ q_{k-2} \\ q_{k-1} \end{pmatrix}$$

The last row ensures that $R(X)$ is given with k coefficients. In terms of elementary operations over $GF(p)$, the complexity of this method is: $k^2 + (k-1)^2$ multiplications and $(k-1)^2 + (k-2)^2$ additions. Furthermore, we can note that if $p < 512$, elementary operations over $GF(p)$ can be implemented with a lookup table.

3 Alternate polynomial representation

Thanks to Lagrange's theorem, we can represent polynomials of degree less than k with their values at k distinct points $\{e_1, e_2, \dots, e_k\}$, i.e., if $A(X)$ is a polynomial of degree at most $k-1$, we denote $ae_i = A(e_i)$ and we represent it with the sequence $(ae_1, ae_2, \dots, ae_k)$ (length k). Unlike the previous approach which uses the polynomial $G(X) = X^k$, we define

$$G(X) = (X - e_1)(X - e_2)\dots(X - e_k), \tag{1}$$

where $e_i \in \{0, 1, \dots, p-1\}$. This clearly implies $p > k$. As we shall see further, $2k$ such distinct points are actually needed, which in turn implies $p > 2k$. The following algorithm computes $A(X) \times B(X) \times G^{-1}(X) \mod N(X)$ for $p > 2k$.

Algorithm 1

Step 1: Define the polynomial $Q(X)$ of degree less than k such that:

$$Q(X) = \left[-A(X) \times B(X) \times N^{-1}(X)\right] \bmod G(X),$$

in other words, we compute in parallel (in $GF(p)$)

$$Q(x) = \left[-A(x) \times B(x) \times N^{-1}(x)\right] \text{ for } x \in \{e_1, e_2, \dots, e_k\}.$$

Step 2: since $[A(X) \times B(X) + Q(X) \times N(X)]$ is a multiple of $G(X)$, we compute $R(X)$ (of degree less than k) such that

$$R(X) = \Big[A(X)\, B(X) + Q(X)\, N(X)\Big] \times G^{-1}(X)$$

In algorithm 1 it is important to note that it is impossible to evaluate $R(X)$ directly as mentioned in step 2. Since we only know the values of the polynomials $A(X)$, $B(X)$, $Q(X)$, $N(X)$ and $G(X)$ for $X \in \{e_1, e_2, \ldots, e_k\}$, it is clear that the sequences representing $[A(X) \times B(X) + Q(X) \times N(X)]$ and $G(X)$ are merely composed of 0. Thus the division by $G(X)$, which actually reduces to the multiplication by $G^{-1}(X)$, has no effect. We address this problem by using k extra values $\{e'_1, e'_2, \ldots, e'_k\}$ where $e'_i \neq e_j$ for all i, j, and by computing $[A(X) \times B(X) + Q(X) \times N(X)]$ and $G(X)$ for those extra values. In the modified algorithm 2, the operation in step 2 is then performed for $X \in \{e'_1, e'_2, \ldots, e'_k\}$.

Algorithm 2

Step 1 Compute $Q(X) = -A(x) \times B(x) \times N^{-1}(x)$ for $x \in \{e_1, e_2, \ldots, e_k\}$ (in parallel),
Step 2 Extend $Q(X)$ for $x \in \{e'_1, e'_2, \ldots, e'_k\}$ using Lagrange interpolation,
Step 3 Compute $R(X)$ in $\{e'_1, e'_2, \ldots, e'_k\}$

$$R(X) = \Big[A(X)\, B(X) + Q(X)\, N(X)\Big] \times G^{-1}(X),$$

Step 4 Extend $R(X)$ back in $\{e_1, e_2, \ldots, e_k\}$ using Lagrange interpolation.

Steps 1 and 3 are fully parallel operations in $GF(p)$. The complexity of algorithm 2 mainly depends on the two polynomial interpolations (steps 2, 4).

3.1 Implementation

In step 1 we compute in $GF(p)$ and in parallel for all i in $\{1, \ldots, k\}$

$$qe_i = -ae_i \times be_i \times n'e_i,$$

where $n'e_i = N^{-1}(e_i)$.

Then in step 2, the extension is performed via Lagrange interpolation:

$$Q(X) = \sum_{i=1}^{k} qe_i \left(\prod_{j=1, j\neq i}^{k} \frac{X - e_j}{e_i - e_j} \right) \tag{2}$$

If we denote $\omega_{t,i} = \prod_{j=1,j\neq i}^{k} \dfrac{e'_t - e_j}{e_i - e_j}$, the extension of $Q(X)$ in $\{e'_1, e'_2, \ldots, e'_k\}$ becomes

$$\begin{pmatrix} qe'_1 \\ qe'_2 \\ \vdots \\ qe'_{k-1} \\ qe'_k \end{pmatrix} = \begin{pmatrix} \omega_{1,1} & \omega_{1,2} & \cdots & \omega_{1,k-1} & \omega_{1,k} \\ \omega_{2,1} & \omega_{2,2} & \cdots & \omega_{2,k-1} & \omega_{2,k} \\ \vdots & & & & \\ \omega_{k-1,1} & \omega_{k-1,2} & \cdots & \omega_{k-1,k-1} & \omega_{k-1,k} \\ \omega_{k,1} & \omega_{k,2} & \cdots & \omega_{k,k-1} & \omega_{k,k} \end{pmatrix} \begin{pmatrix} qe_1 \\ qe_2 \\ \vdots \\ qe_{k-1} \\ qe_k \end{pmatrix} \tag{3}$$

Operations in step 3 are performed in parallel for i in $\{1, \ldots, k\}$. We compute in $GF(p)$

$$re'_i = (ae'_i * be'_i + qe'_i * ne'_i) * \zeta_i$$

where

$$\zeta_i = \left[\prod_{j=1}^{k} (e'_i - e_j) \right]^{-1} = [G(e'_i)]^{-1} \bmod p.$$

It is easy to remark that $G(e'_i) \neq 0$ for i in $\{1, \ldots, k\}$.

At the end of step 3, the polynomial $R(X)$ of degree less than k is defined by its k values in $GF(p)$ for $X \in \{e'_1, e'_2, \ldots, e'_k\}$.

In step 4 we extend $R(X)$ back in e. As in step 2 we define

$$\omega'_{t,i} = \prod_{j=1, j\neq i}^{k} \frac{e_t - e'_j}{e'_i - e'_j},$$

and we compute

$$\begin{pmatrix} re_1 \\ re_2 \\ \vdots \\ re_{k-1} \\ re_k \end{pmatrix} = \begin{pmatrix} \omega'_{1,1} & \omega'_{1,2} & \cdots & \omega'_{1,k-1} & \omega'_{1,k} \\ \omega'_{2,1} & \omega'_{2,2} & \cdots & \omega'_{2,k-1} & \omega'_{2,k} \\ \vdots & & & & \\ \omega'_{k-1,1} & \omega'_{k-1,2} & \cdots & \omega'_{k-1,k-1} & \omega'_{k-1,k} \\ \omega'_{k,1} & \omega'_{k,2} & \cdots & \omega'_{k,k-1} & \omega'_{k,k} \end{pmatrix} \begin{pmatrix} re'_1 \\ re'_2 \\ \vdots \\ re'_{k-1} \\ re'_k \end{pmatrix} \tag{4}$$

Complexity In terms of elementary operations, the complexity of this method is $2k^2$ multiplications by a constant and $2k(k-1)$ additions in $GF(p)$.

4 Example

Let us first define the constant parameters. We consider the finite field $GF(23^5)$ according to the irreducible polynomial of degree 5: $N(x) = x^5 + 2x + 1$, the two sets of points $e = \{2, 4, 6, 8, 10\}$ and $e' = \{3, 5, 7, 9, 11\}$, the interpolation matrices needed in steps 2 and 4:

$$\omega = \begin{pmatrix} 8 & 9 & 7 & 11 & 12 \\ 12 & 17 & 14 & 2 & 2 \\ 2 & 2 & 14 & 17 & 12 \\ 12 & 11 & 7 & 9 & 8 \\ 8 & 18 & 22 & 19 & 3 \end{pmatrix} \quad \text{and} \quad \omega' = \begin{pmatrix} 3 & 19 & 22 & 18 & 8 \\ 8 & 9 & 7 & 11 & 12 \\ 12 & 17 & 14 & 2 & 2 \\ 2 & 2 & 14 & 17 & 12 \\ 12 & 11 & 7 & 9 & 8 \end{pmatrix},$$

and the vector $\zeta = (16, 1, 22, 7, 12)$ used in step 3.

Given the two elements $A(X)$ and $B(X)$ of $GF(23^5)$:

$$A(x) = 2x^4 + x + 3 \qquad B(x) = x^2 + 5x + 4,$$

we aim at computing $R(X) = A(X)B(X)G^{-1}(X) \bmod N(X)$ in the evaluated form $re = (R(e_1), R(e_2), \ldots, R(e_k))$.

We evaluate A, B and N at each value of e and e':

$$ae = (14, 13, 2, 15, 3) \quad \text{and} \quad ae' = (7, 16, 5, 1, 17),$$

$$be = (18, 17, 1, 16, 16) \quad \text{and} \quad be' = (5, 8, 19, 15, 19),$$

$$ne = (14, 21, 15, 10, 17) \quad \text{and} \quad ne' = (20, 8, 9, 4, 5);$$

and we compute the vector

$$n'e = (5, 11, 20, 7, 19).$$

In step 1 of the algorithm we compute

$$qe = (5, 7, 6, 22, 8),$$

and we extend it in step 2 from e to e' (eq. (3)):

$$qe' = (0, 1, 3, 4, 4).$$

Now in step 3, we evaluate in parallel for each value of e':

$$re' = (8, 21, 16, 10, 22),$$

and we interpolate it back (eq. (4)) to obtain the final result in e:

$$re = (4, 3, 5, 3, 15).$$

It is easy to verify that the results re and re' are correct by evaluating

$$R(X) = A(X)B(X)G^{-1}(X) \bmod N(X) = 3X^4 + 17X^3 + 11X^2 + 6X + 17$$

at each points of e and e'.

5 Discussions

5.1 Simplified architecture

A major advantage of this method is that the matrices in (3) and (4) do not depend on the inputs. Thus, all the operations reduce to multiplications by constants which significantly simplify the hardware implementation. Moreover, in the example presented in section 4, we have detected symmetries between the two matrices that can also contribute to a simplified architecture.

Lemma 1. *Let us denote*

$$\omega_{i,j} = \prod_{m=1, m\neq j}^{k} \frac{2i+1-2m}{2j-2m} \quad \textit{and} \quad \omega'_{i,j} = \prod_{m=1, m\neq j}^{k} \frac{(2i+1-(2m+1))}{(2j+1-(2m+1))}, \tag{5}$$

for $i, j \in \{1, \ldots, k\}$. *Then for every* $i, j \in \{1, \ldots, k\}$ *we have*

$$\omega_{i,j} = \omega'_{k+1-i,k+1-j}.$$

In other words equation (4) can be implemented with the same matrix than eq. (3), by simply reversing the order of the elements of the vectors re *and* re'*:*

$$\begin{pmatrix} re_k \\ re_{k-1} \\ \vdots \\ re_2 \\ re_1 \end{pmatrix} = \begin{pmatrix} \omega_{1,1} & \omega_{1,2} & \ldots & \omega_{1,k-1} & \omega_{1,k} \\ \omega_{2,1} & \omega_{2,2} & \ldots & \omega_{2,k-1} & \omega_{2,k} \\ \vdots & & & & \\ \omega_{k-1,1} & \omega_{k-1,2} & \ldots & \omega_{k-1,k-1} & \omega_{k-1,k} \\ \omega_{k,1} & \omega_{k,2} & \ldots & \omega_{k,k-1} & \omega_{k,k} \end{pmatrix} \begin{pmatrix} re'_k \\ re'_{k-1} \\ \vdots \\ re'_2 \\ re'_1 \end{pmatrix} \tag{6}$$

The proof of Lemma 1 is given in Appendix A.

Lemma 2. *Under the same conditions than those of equations (5) in Lemma 1 ; then, for all* $i \in \{2, \ldots, k\},\ j \in \{1, \ldots, k\}$ *we have the identity*

$$\omega'_{i,j} = \omega_{i-1,j}.$$

$$\begin{pmatrix} \omega'_{1,1} & \omega'_{1,2} & \ldots & \omega'_{1,k-1} & \omega'_{1,k} \\ \omega'_{2,1} & \omega'_{2,2} & \ldots & \omega'_{2,k-1} & \omega'_{2,k} \\ \vdots & & & & \\ \omega'_{k-1,1} & \omega'_{k-1,2} & \ldots & \omega'_{k-1,k-1} & \omega'_{k-1,k} \\ \omega'_{k,1} & \omega'_{k,2} & \ldots & \omega'_{k,k-1} & \omega'_{k,k} \end{pmatrix} = \begin{pmatrix} \omega_{k,k} & \omega_{k,k-1} & \ldots & \omega_{k,2} & \omega_{k,1} \\ \omega_{1,1} & \omega_{1,2} & \ldots & \omega_{1,k-1} & \omega_{1,k} \\ \vdots & & & & \\ \omega_{k-2,1} & \omega_{k-2,2} & \ldots & \omega_{k-2,k-1} & \omega_{k-2,k} \\ \omega_{k-1,1} & \omega_{k-1,2} & \ldots & \omega_{k-1,k-1} & \omega_{k-1,k} \end{pmatrix}$$

The proof of Lemma 2 is given in Appendix B.

Remarks These two lemmas point out symmetry properties of the matrices that mainly depend on the choice made in the example for the points of e and e'. The can be taken into account to improve the hardware architecture. Other choice of points could be more interesting and could result in very attractive chip design solutions. It is currently a work in progress in our team.

5.2 Cryptographic context

In ECC, the main operation is the addition of two points of an elliptic curve over a finite field. Hardware implementation of elliptic curves cryptosystems thus requires efficient operators for additions, multiplications and divisions. Since division is usually a complex operation, we use homogeneous coordinates to bypass this difficulty (only one division is needed at the very end of the algorithm).

Thus the only operations are addition and multiplication in $GF(p)$. The cost of an addition over $GF(p)$ is no more than p Full-Adders. Actually we do not have to reduce modulo p after each addition. We only subtract p from the result of the last addition if it is greater than $2^{\lceil log_2(p) \rceil}$ (we recall that p is odd). In other words we just have to check one bit after each addition. The exact value is only needed for the final result.

In ECC protocols, additions chains of points of an elliptic curve are needed. In homogeneous coordinates, those operations consist in additions and multiplications over $GF(p^k)$. Only one division is needed at the end and it can be performed in the Lagrange representation using the Fermat-Euler theorem which states that for all non zero value x in $GF(p^k)$, then $x^{p^k-1} = 1$. Hence we can compute the inverse of x by computing x^{p^k-2} in $GF(p^k)$.

It is also advantageous to use a polynomial equivalent to the Montgomery notation during the computations. We consider the polynomials in the form $A'(X) = A(X) \times G(X) \bmod N(X)$ instead of $A(X)$. It is clear that adding two polynomials given in this notation gives the result in the same notation, and for the product, since $\mathrm{Mont}(A, B, N) = A(X) \times B(X) \times G^{-1}(X) \bmod N(X)$, we have $\mathrm{Mont}(A', B', N) = A'(X) \times B'(X) \times G^{-1}(X) \bmod N(X) = A(X) \times B(X) \times G(X) \bmod N(X)$.

6 Conclusion

Recent works from Bailey and Paar have shown that it is possible to obtain more efficient software implementation over $GF(p^k)$ than over $GF(2^k)$ or $GF(p)$. In this article we have presented a new modular multiplication algorithm over the finite extension field $GF(p^k)$, for $p > 2k$, which is highly parallelizable and well adapted to hardware implementation. Our algorithm is particularly interesting for ECC since it seems that there exists less nonsingular curves over $GF(p^k)$ than over $GF(2^k)$. Finding "good" curves for elliptic curve cryptography would then be easier. Furthermore, under the condition "k is a power of a prime number $q \geqslant 11$", the primality condition on k required for the fields $GF(2^k)$ could be released in the case $GF(p^k)$. This could result in a wider choice of curves than in the case $p = 2$. This method can be extended to finite fields on the form $GF(2^{nm})$, where $2^n > 2m$. Fields of this form can be useful for the recent tripartite Diffie-Hellamn key exchange algorithm [12] or the short signature scheme [13] which require an efficient arithmetic over $GF(p^{kl})$, where $6 < k \leqslant 15$ and l is a prime number greater than 160. In this case $p = 2^n$ is no longer a prime number which forces us to choose the values of e and e' in $GF(2^n)^*$.

References

1. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation **48** (1987) 203–209
2. Koblitz, N.: A Course in Number Theory and Cryptography. second edn. Volume 114 of Graduate texts in mathematics. Springer-Verlag (1994)

3. Koblitz, N.: Algebraic aspects of cryptography. Volume 3 of Algorithms and computation in mathematics. Springer-Verlag (1998)
4. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22** (1976) 644–654
5. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA (1997)
6. Yanik, T., Savaş, E., Ç. K. Koç: Incomplete reduction in modular arithmetic. IEE Proceedings: Computers and Digital Technique **149** (2002) 46–52
7. Ç. K. Koç, Acar, T.: Montgomery multiplication in GF(2^k). Designs, Codes and Cryptography **14** (1998) 57–69
8. Halbutoğullari, A., Ç. K. Koç: Parallel multiplication in GF(2^k) using polynomial residue arithmetic. Designs, Codes and Cryptography **20** (2000) 155–173
9. Paar, C., Fleischmann, P., Roelse, P.: Efficient multiplier architectures for galois fields GF(2^{4n}). IEEE Transactions on Computers **47** (1998) 162–170
10. Bailey, D., Paar, C.: "efficient arithmetic in finite field extensions with application in elliptic curve cryptography. Journal of Cryptology **14** (2001) 153–176
11. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation **44** (1985) 519–521
12. Joux, A.: A one round protocol for tripartite Diffie-Hellman. In: 4th International Algorithmic Number Theory Symposium (ANTS-IV. Volume 1838 of Lecture Notes in Computer Science., Springer-Verlag (2000) 385–393
13. Boneh, D., Shacham, H., Lynn, B.: Short signatures from the Weil pairing. In: proceedings of Asiacrypt'01. Volume 2139 of Lecture Notes in Computer Science., Springer-Verlag (2001) 514–532

A Proof of Lemma 1

We are going to rearrange each part of the equality, to make appear the identity.

We first focus on the right part of the identity:

$$\begin{aligned}\omega'_{k+1-i,k+1-j} &= \textstyle\prod_{m=1,m\neq k+1-j}^{k} \dfrac{(2(k+1-i)-(2m+1))}{(2(k+1-j)+1-(2m+1))} \\ &= \textstyle\prod_{m=1,m\neq k+1-j}^{k} \dfrac{(-2i+2(k+1-m)-1)}{(-2j+1+2(k+1-m)-1)}\end{aligned}$$

Here we have just changed the place of $k+1$ in each term of the product. Next just by simplifying each fraction by -1, and factorizing all the 2 in the denominators, we get:

$$\begin{aligned}\omega'_{k+1-i,k+1-j} &= 2^{1-k} \textstyle\prod_{m=1,m\neq k+1-j}^{k} \dfrac{(2i-2(k+1-m)+1)}{(j-(k+1-m))} \\ &= 2^{1-k} \textstyle\prod_{m=1,m\neq j}^{k} \dfrac{(2i+1-2m)}{(j-m)}\end{aligned}$$

Here we have changed the indices $m \leftarrow k+1-m$.

We now do the same procedure with the left term.

$$\omega_{i,j} = \prod_{m=1,m\neq j} \frac{2i+1-2m}{2j-2m}$$

We factorize the 2 in the denominators:

$$\omega_{i,j} = 2^{1-k} \prod_{m=1,m\neq j} (\frac{2i+1-2m}{j-m})$$

We can then conclude that the new expressions of $\omega_{i,j}$ and $\omega'_{k+1-i,k+1-j}$ are the same.

B Proof of Lemma 2

Here again, this is proved with only simple manipulations on the coefficients. Let us begin with the expression of $\omega_{i-1,j}$

$$\begin{aligned}\omega_{i-1,j} &= \textstyle\prod_{m=1,m\neq j}^{k} \dfrac{2(i-1)+1-2m}{2j-2m} \\ &= \textstyle\prod_{m=1,m\neq j}^{k} \dfrac{2i-1-2m}{2j+1-1-2m}\end{aligned}$$

Here we have just used the equalities $2(i-1)+1 = 2i-1$ in the numerators and $0 = 1-1$ in the denominators. So rearranging, each factor of the product, gives:

$$\begin{aligned}\omega_{i-1,j} &= \textstyle\prod_{m=1,m\neq j}^{k} \frac{2i-(2m+1)}{((2j+1)-(2m+1))} \\ &= \omega'_{i,j}\end{aligned}$$