Using an RSA Accelerator for Modular Inversion

Martin Seysen

Giesecke & Devrient GmbH, Prinzregentenstraße 159,
D-81677 Munich, Germany
Martin.Seysen@de.gi-de.com

Abstract. We present a very simple new algorithm for modular inversion. Modular inversion can be done by the extended Euclidean algorithm. We substitute the extended Euclidean algorithm by a standard (non-extended) Euclidean algorithm that works on integers of approximately double the length of the modulus. This substitution can be very useful on smart card coprocessors, since in some cases computations with longer numbers than necessary can be done at no extra cost. Many smart card coprocessors have been designed for the RSA algorithm of, say, 1024 bits length. On the other hand, elliptic curve algorithms work with much smaller numbers, and modular inversion is a much more important primitive in elliptic curve cryptography than in RSA cryptography. On one smart card coprocessor the new algorithm is more than twice as fast as the classical algorithm.

Key Words:
smart card coprocessor, modular inversion, Euclidean algorithm.

1 Introduction

When public key cryptography was first used in low-cost devices such as smart cards, it turned out that the standard CPU of such a device is too slow to perform the necessary cryptographic operations. At that time algorithms that work in the multiplicative group modulo a long integer such as RSA or Diffie-Hellman were the most popular public key algorithms. Several coprocessors for performing operations on long integers have been designed. Most of them have been optimised for modular multiplications of $512, \ldots, 1024$ bit length, as required for public key algorithms in the early nineties of the last century. A variety of different modular multiplication techniques has been used in such coprocessors, see [12, 14, 15, 19]. For an overview of modular multiplication techniques see [8].

Depending on the architecture, such a coprocessor performs either operations modulo numbers of a fixed (maximum) bit length, or the bit length is limited only by memory resources. Since the required key size for public key systems increases during the years (see e.g. [10]), coprocessors must deal with larger

numbers than they have been designed for. Solutions for this problem have been given in [4, 13, 2].

When elliptic curve cryptography became more popular on smart cards in the late nineties, implementers had to deal with the opposite problem. Here the key size is typically in the range of $160, \ldots, 256$ bits, which is much smaller than the key size in the RSA algorithm. This means that coprocessors have to deal with much smaller numbers than they have been designed for. Another problem is that some coprocessors are optimised for modular multiplication, as required for the RSA algorithm. But other operations such as modular addition, subtraction and inversion are more important in EC operations than in the RSA algorithm. Here we mainly consider elliptic curves over GF(p).

This paper deals with the modular inversion on smart card coprocessors. The oldest algorithm for computing the modular inverse is the extended Euclidean algorithm for computing the greatest common divisor (gcd), see e.g. [8]. A binary algorithm for computing the gcd has been proposed in [18]. Modular inversion techniques have been investigated under a variety of aspects. The classical gcd algorithms have been improved and optimised for large numbers on a CPU with a fixed word length, see e.g. [6, 9, 16, 17]. A hardware-optimised modular inversion algorithm has been proposed in [11]. Modular inversion algorithms that avoid the computation of the greatest common divisor have been proposed in [7].

While modular multiplication is hardware supported by a coprocessor on many smart cards, modular inversion usually has to be coded in software. On some CPUs a modular inversion may be about 100 times slower than a modular multiplication.

Therefore it is crucial to use as little modular inversions as possible in EC operations on smart cards. Techniques for reducing the number of modular inversions in EC operations are well-known, see e.g. [1] or [5] for an overview. Here the basic idea is to avoid modular inversions by representing the points on a curve in projective or Jacobian projective co-ordinates instead of affine co-ordinates. We can also used mixed co-ordinate systems to reduce the number of arithmetic operations. These techniques can be combined with windowing methods to reduce the number of EC operations. Depending on the details of the implementation, more inversions may be necessary in this case, see [3].

One of the most well-known EC algorithms is the ECDSA algorithm. Note that apart from the elliptic curve operations, the ECDSA signing algorithm requires another modular inversion operation modulo the group order.

When running the ECDSA signing algorithm with $160, \ldots, 192$ bit length on a smart card coprocessor, then up to about 20 percent of the run time may be spent for modular inversions.

Therefore any significant speedup of the modular inversion algorithm on a smart card coprocessor has a non-negligible effect on the run time of the ECDSA signing algorithm.

We propose a very simple new modular inversion algorithm that is specially suited for a coprocessor optimised for RSA cryptography. The new algorithm performs about half the number of operations compared to the classical extended

Euclidean algorithm, but it works with double-length numbers. As we shall see, the speed of the new algorithm is more than doubled compared to the standard implementation of the modular inversion on the Infineon SLE66CX322P CPU. The main reason for this speedup is the fact that basic double-length integer operations such as addition, subtraction and shifting, are not much more expensive than single-length operations on this CPU for key sizes of $160, \ldots, 256$ bits.

The new algorithm may also be useful on other smart card CPUs, since many of these CPUs combine a highly optimised coprocessor for long-integer arithmetic with a main processor of much less performance. Therefore instructions such as register switches, loop control, pointer management or data transfer between main processor and coprocessor, (called glue instructions in [7],) may take up a considerable part of the run time of a modular inversion algorithm. Obviously, less glue instructions are necessary, when fewer operations are performed on longer integers.

2 Modular Inversion with a Non-extended Euclidean Algorithm

In this paper we will show the correctness of the following modular inversion algorithm:

Algorithm NINV
(Modular inversion with a non-extended Euclidean algorithm)

`Input` *Integers* $u \geq 0$, $v > 1$, *and an arbitrary extension factor* f *with* $f > 2v$.

`Output` *Modular inverse* $x = u^{-1} \pmod{v}$ *with* $-v < x < v$, *or an error if* u *is not invertible modulo* v.

[1] `Put` $U = fu + 1, V = fv$.

[2] `While` $V \geq f + v$ `do`
$\{T = V, V = U \bmod V, U = T\}$.

[3] `If` $V > f - v$ `then return` $V - f$ `and stop` ,
`else return` *"error"* `and stop` .

Throughout the paper we write $u \bmod v$ for the integer x satisfying $x = u \pmod{v}$, $0 \leq x < v$.

For the analysis of Algorithm NINV we may rephrase Step 2 as follows:

$$U_0 = U = fu + 1\ ,\ V_0 = V = fv\ ;$$
$$U_{i+1} = V_i\ , \quad V_{i+1} = U_i \bmod V_i\ ; \quad \text{for } i \geq 0\ ,\ V_i > 0\ . \tag{1}$$

Note that this is exactly the process of the Euclidean algorithm applied to U and V. The number of Euclidean steps in Algorithm NINV is the about the same as the number of Euclidean steps in the standard Euclidean algorithms applied to u and v. See [8] for an analysis of the number of Euclidean steps required in

the Euclidean algorithm. The extension factor f can be chosen as a power of two such that the bit length of the numbers U and V is about twice the bit length of $\max(u, v)$. So Algorithm NINV requires roughly the same number of operations as the standard (non-extended) Euclidean algorithm, but it has to work with integers of double length. On the other hand the overhead for the 'extended' part of the Euclidean algorithm is saved. Depending on the details of the smart card CPU, this may lead to considerable saving of glue instructions.

Before showing the correctness of the algorithm, we first give some motivation why the algorithm is expected to work. The Euclidean algorithm, when applied to u and v returns a number $d = \gcd(u, v)$. It is well known that the algorithm can be extended to compute integers λ, μ, $|\lambda| < v$, with $d = \lambda u - \mu v$. In case $d = 1$ the number λ is just the modular inverse we are looking for. When we apply the Euclidean process to $\tilde{U} = fu$ and $V = fv$ instead of u and v, we obtain exactly the same sequences of quotients in the modular reduction operations in both cases, and we also obtain a linear combination $\gcd(\tilde{U}, V) = fd = \lambda\tilde{U} - \mu V$ with the same values λ, μ as above. Now we introduce a small perturbation by replacing $\tilde{U}$ by $U = \tilde{U} + 1$. Assume for a moment that this perturbation does not change the sequence of quotients in the Euclidean process. Then a remainder $\lambda U - \mu V = fd + \lambda$ would occur in the sequence (V_i) of remainders in the Euclidean process, and in case $d = 1$ we could simply read the modular inverse λ from a remainder of size $\approx f$. Although the above assumption is in general not true, the construction of Algorithm NINV is based on this idea. More specifically, the following theorem proves the correctness of Algorithm NINV.

Theorem 1 *In case* $\gcd(u, v) = 1$ *there is an integer* i *such that* $V_{i-1} > 2f - v$, $f + v > V_i > f - v$ *and* $(V_i - f) \cdot u = 1 \pmod{v}$ *hold. Otherwise every* V_i *satisfies either* $V_i > 2f - v$ *or* $V_i \leq \frac{v}{2}$.

The proof of the theorem uses continued fractions. Some basic facts about continued fractions are stated in the next section. Theorem 1 is proved in section 4.

The relation between the standard extended Euclidean algorithm and Algorithm NINV can be stated as follows. The standard algorithm computes integers $u_i, v_i, \lambda_i, \lambda'_i$ with $\lambda_i u \equiv u_i$, $\lambda'_i u \equiv v_i \pmod{v}$, starting with $u_0 = u$, $v_0 = v$, $\lambda_0 = 1$, $\lambda'_0 = 0$; and it stops when arriving at some v_i with $v_i = \gcd(u, v)$. Algorithm NINV stores the numbers $U_i = fu_i + \lambda_i$ and $V_i = fv_i + \lambda'_i$ in two double-length variables instead. Note that some of the perturbations λ_i, λ'_i can be negative and may spoil the Euclidean quotients. This means that we may have $\lfloor U_i/V_i \rfloor \neq \lfloor u_i/v_i \rfloor$ in some cases, regardless of the size of f. Therefore we have to modify the analysis of the standard extended Euclidean algorithm to obtain a proof of Theorem 1.

3 Continued Fractions

Algorithm NINV can most easily be analysed in terms of continued fractions. We take notation for and standard facts about continued fractions from [8]. A

continued fraction $/\!/x_1, x_2, \ldots, x_{n-1}, x_n/\!/$ is defined by:

$$/\!/x_1, x_2, \ldots, x_{n-1}, x_n/\!/ = 1/(x_1 + 1/(x_2 + 1/(\ldots(x_{n-1} + 1/x_n)\ldots))).$$

Continued fractions are closely related to the so-called continuant polynomials $K_n(x_1, \ldots, x_n)$ defined by:

$$K_n(x_1, \ldots, x_n) = \begin{cases} 1 & \text{If } n = 0; \\ x_1 & \text{If } n = 1; \\ x_1 K_{n-1}(x_2, \ldots, x_n) + K_{n-2}(x_3, \ldots, x_n) & \text{If } n > 1. \end{cases}$$

Then the following explicit formulas hold for continued fractions:

$$/\!/x_1, x_2, \ldots, x_n/\!/ = \frac{K_{n-1}(x_2, \ldots, x_n)}{K_n(x_1, x_2, \ldots, x_n)} \tag{2}$$

$$x_0 + /\!/x_1, x_2, \ldots, x_n/\!/ = \frac{K_{n+1}(x_0, x_1, \ldots, x_n)}{K_n(x_1, x_2, \ldots, x_n)} \tag{3}$$

Every real number X has a (regular) continued fraction expansion defined as follows: Let $A_0 = \lfloor X \rfloor, X_0 = X - A_0$, and for all $n \geq 0$ such that $X_n \neq 0$ define:

$$A_{n+1} = \lfloor 1/X_n \rfloor, X_{n+1} = 1/X_n - A_{n+1}. \tag{4}$$

For a rational number X there is an integer n such that A_{n+1} is not defined; and the continued fraction expansion of X is:

$$X = A_0 + /\!/A_1, \ldots, A_n/\!/. \tag{5}$$

For an irrational number X, an infinite continued fraction expansion can be defined.

For a given X, we define quantities p_i, q_i by:

$$\begin{aligned} p_{-1} = 1, q_{-1} = 0, p_0 = A_0, q_0 = 1, \\ p_{i+1} = A_i p_i + p_{i-1},\ q_{i+1} = A_i q_i + q_{i-1} \quad ;\ i = 0, \ldots, n-1\ . \end{aligned} \tag{6}$$

By induction over i we easily obtain:

$$p_i = K_{i+1}(A_0, \ldots, A_i) \quad ;\ i = 0, \ldots, n\ ; \tag{7}$$

$$q_i = K_i(A_1, \ldots, A_i) \quad ;\ i = 0, \ldots, n\ ; \tag{8}$$

$$\frac{p_i}{q_i} = A_0 + /\!/A_1, \ldots, A_i/\!/ \quad ;\ i = 1, \ldots, n\ . \tag{9}$$

For a number X, assume that in its continued fraction expansion $A_0 + /\!/A_1, A_2, \ldots/\!/$ the term A_{i+1} exists. Then the following facts are well-known, see e.g. [8].

The number X lies between p_i/q_i and p_{i+1}/q_{i+1}, and we have:

$$q_{i+1} > q_i \quad ; \quad |p_i q_{i+1} - p_{i+1} q_i| = 1\ ; \tag{10}$$

$$\left| X - \frac{p_i}{q_i} \right| \leq \frac{1}{q_i q_{i+1}} < \frac{1}{q_i^2}\ . \tag{11}$$

The last fact means that the rational numbers p_i/q_i are very close rational approximations for the number X. They are called the convergents of (the regular continued fraction expansion for) X.

The following classical result is due to A. M. Legendre:

Theorem 2 *If a rational number p/q with $p \in \mathbb{Z}, q \in \mathbb{N}$, satisfies*

$$\left| X - \frac{p}{q} \right| \leq \frac{1}{2q^2} \ ,$$

then p/q is a convergent of the regular continued fraction expansion for X.

We still need another fact about continued fractions:

Proposition 3 *Assume that μ_1/λ_1 and μ_2/λ_2 are two convergents of the regular continued fraction expansion for X satisfying $|\mu_1\lambda_2 - \mu_2/\lambda_1| = 1$. Then these two convergents are either adjacent in the sequence of convergents for X, or there is exactly one more convergent $(\mu_2 - \mu_1)/(\lambda_2 - \lambda_1)$ between them.*

Sketch Proof

W.l.o.g. we assume $\lambda_1 < \lambda_2$. The Diophantine equation $|\mu_2 x - \lambda_2 y| = 1$ has exactly two solutions (x, y) satisfying $0 \leq x < \lambda_2$, and we easily identify these two solutions as (λ_1, μ_1) and (λ_3, μ_3), with $\lambda_3 = \lambda_2 - \lambda_1$, $\mu_3 = \mu_2 - \mu_1$. Assume that μ_1/λ_1 and μ_2/λ_2 are not adjacent. Then by (10), μ_3/λ_3 is the convergent preceding μ_2/λ_2. Now let μ_4/λ_4 be the convergent preceding μ_3/λ_3. By (6) there is an integer $A \geq 1$ with $\lambda_2 = A\lambda_3 + \lambda_4$. Hence

$$\lambda_4 \leq \lambda_2 - \lambda_3 = \lambda_1 \ .$$

But since by assumption μ_4/λ_4 does not precede μ_1/λ_1 in the series of convergents, these two quantities are equal.
□

4 Proof of Theorem 1

Applying the Euclidean Algorithm to U and V as stated in (1) corresponds in a natural way to the regular continued fraction expansion of the rational number $X = U/V$.

Define:

$$\bar{A}_i = \lfloor U_i/V_i \rfloor \ ,$$
$$\bar{X}_i = U_i/V_i - \bar{A}_i = (U_i \bmod V_i)/V_i = V_{i+1}/U_{i+1} \ ; \quad i \geq 0, V_i > 0 \ . \qquad (12)$$

Comparing (4) with (1) and (12) we see that the quantities $\bar{A}_i, \bar{X}_i$ defined in (12) are exactly the numbers A_i, X_i that appear in the continued fraction expansion of $X = U/V$.

Regarding the quantities p_i and q_i defined by (6), and using induction over i, we obtain from (1) and (12):

$$q_i U - p_i V = (-1)^i V_{i+1} \quad ; \; i \geq -1 \; . \tag{13}$$

Define $d = \gcd(u, v)$. There are coprime integers λ, μ with

$$\lambda u - \mu v = d \; . \tag{14}$$

We now assume that $u \neq 0 \pmod{v}$ holds in Algorithm NINV. Otherwise we have $U \bmod V = 1$ and the algorithm terminates with an error as expected.

Since $v \neq 0, u \neq 0 \pmod{v}$, we have $0 < d < v$. Hence $\lambda \neq 0 \bmod v$ holds and λ and μ cannot have opposite signs. So we have $\mu/\lambda \geq 0$ in all cases. From (1) we obtain:

$$\left| \frac{|\mu|}{|\lambda|} - \frac{U}{V} \right| = \left| \frac{\mu}{\lambda} - \frac{U}{V} \right| = \left| \frac{fd + \lambda}{f\lambda v} \right| \quad ; \; \lambda \neq 0 \pmod{v} \; ; \; \frac{\mu}{\lambda} \geq 0 \; . \tag{15}$$

In (14) we may replace λ and μ by $\lambda + kv/d$ and $\mu + ku/d$ for any integer k.

Our goal is now to find values λ and μ with $|\mu/\lambda - U/V| < 1/(2\lambda^2)$ so that we may apply Theorem 2. A sufficient condition for this is:

$$-\frac{1}{2}df + \frac{1}{2}\sqrt{(df)^2 - 2fv} < \lambda < -\frac{1}{2}df + \frac{1}{2}\sqrt{(df)^2 + 2fv} \quad ; \; f \geq 2v \; . \tag{16}$$

We easily check that length of the feasible interval for λ given by (16) is always greater than v/d. So we can always find integers $\lambda, \mu, \mu/\lambda \geq 0$ satisfying (14) and $|\mu/\lambda - U/V| < 1/(2\lambda^2)$. Thus by Theorem 2 we obtain:

Lemma 4 *There exist coprime integers λ, μ satisfying $\lambda u - \mu v = \gcd(u, v)$ such that $|\mu/\lambda|$ is a convergent of the regular continued fraction expansion of U/V.*

Let λ_{max} be the highest possible value for λ that satisfies Lemma 4. We want to calculate an upper and a lower bound for λ_{max}. By (11) and (15), λ_{max} must satisfy:

$$\left| \frac{fd + \lambda_{max}}{f\lambda_{max} v} \right| < \frac{1}{\lambda_{max}^2} \; .$$

This implies:

$$\lambda_{max} < v/d \; . \tag{17}$$

On the other hand, every interval of length at least v/d must contain an integer solution λ to the Dipohantine equation $\lambda u - \mu v = d$. Since the feasible interval for λ given in (16) is sufficiently long, we conclude:

$$\lambda_{max} > \left(-\frac{1}{2}df + \frac{1}{2}\sqrt{(df)^2 + 2fv} \right) - \frac{v}{d} \; > \; -\frac{v}{2d} - \frac{v^2}{4d^3 f} \; > \; -\frac{5v}{8d} \; . \tag{18}$$

Now we choose once and for all the convergent μ/λ satisfying Lemma 4 such that λ is maximal, i.e. $\lambda = \lambda_{max}$.

We have $|u/v - U/V| = 1/V < 1/(2v^2)$, so that by Theorem 2 the rational number $(u/d)/(v/d)$ is a convergent for U/V too. Furthermore, we have $\lambda u/d - \mu v/d = 1$. Thus Proposition 3 implies:

Lemma 5 *Either* $\frac{|\mu|}{|\lambda|}, \frac{u/d}{v/d}$ *or* $\frac{|\mu|}{|\lambda|}, \frac{u/d-|\mu|}{v/d-|\lambda|}, \frac{u/d}{v/d}$ *are adjacent convergents of the regular continued fraction expansion of* U/V.

For the subsequent convergents p_i/q_i given by Lemma 5, we can now use (13) to calculate the corresponding values V_{i+1}; and we will use (17) and (18) to obtain upper and lower bounds for the values V_{i+1}. These values are part of the sequence V_i defined by (1). The result is given in Table 1. Note that the second entry in Table 1 corresponding to the convergent $\frac{u/d-|\mu|}{v/d-|\lambda|}$ may or may not be present in the sequence of convergents.

Table 1. Bounds for the intermediate results $V_{i+1} = (-1)^i(q_iU - p_iV)$ depending on the convergents p_i/q_i

numerator p_i	denominator q_i	V_{i+1}	lower bound for V_{i+1}	upper bound for V_{i+1}
$\|\mu\|$	$\|\lambda\|$	$fd+\lambda$	$fd-\frac{5}{8}v/d$	$fd+v/d$
$u/d-\|\mu\|$	$v/d-\|\lambda\|$	$fd+\lambda-v/d$	$fd-\frac{13}{8}v/d$	fd
u/d	v/d	v/d	v/d	v/d

Proof of Theorem 1
Case 1: $d = \gcd(u, v) > 1$
In this case Table 1 shows that no V_i with $\frac{v}{2} \le V_i \le 2f - \frac{13}{16}v$ exists.
Case 2: $d = 1$
The first entry of Table 1 shows that a V_{i+1} with $f - \frac{5}{8}v < V_{i+1} < f + v$ and $V_{i+1} = f + \lambda$ exists. From (14) we conclude that λ is a modular inverse of u modulo v. Let V_j be the entry of the sequence $V_{i+1}, i = 0, 1, \ldots$ corresponding to the first entry of the table. We still have to show that the entry V_{j-1} preceding V_j is sufficiently large. By (1) there is an integer $A \ge 1$ such that we have:

$$V_{j-1} = AV_j + V_{j+1} \ . \tag{19}$$

Here V_{j+1} corresponds either to entry two or three of Table 1.
Case 2a: V_{j+1} corresponds to entry two of Table 1.

Then there is exactly one more convergent between the convergents $|\mu/\lambda|$ and u/v of U/V. So we conclude from (13) that $|\lambda|U - |\mu|V$ and $vU - uV = v > 0$ have the same sign. By (1) and (14) we have $\lambda U - \mu V = f + \lambda > 0$. Thus λ must be positive in this case. Then by (19) we have:

$$V_{j-1} \ge V_j + V_{j+1} = 2f + 2\lambda - v > 2f - v.$$

Case 2b: V_{j+1} corresponds to entry three of Table 1 and $A > 1$ holds in (19).

Then we have:

$$V_{j-1} \ge 2V_j + V_{j+1} \ge 2f + 2\lambda + v > 2f - v \ .$$

Case 2c: V_{j+1} corresponds to entry three of Table 1 and $A = 1$ holds in (19).

In this case we will show that the algorithm outputs V_{j-1} and that V_{j-2} is sufficiently large. Since $|\mu/\lambda|$ and u/v are adjacent convergents of U/V, we conclude from (13) that $|\lambda|U - |\mu|V$ and $vU - uV = v > 0$ have opposite sign. By (1) and (14) we have $\lambda U - \mu V = f + \lambda > 0$. Thus λ must be negative in this case. Now we compute:

$$V_{j-1} = V_j + V_{j+1} = f + \lambda + v < f + v \; ;$$
$$V_{j-2} \geq V_{j-1} + V_j = 2f + 2\lambda + v > 2f - v \; .$$

From this we see that $V_{j-1} - f = \lambda + v$ is a modular inverse of u modulo v of appropriate size, and that the predecessor V_{j-2} of V_{j-1} is sufficiently large. □

5 Implementation Results

Algorithm NINV has been implemented on the Infineon SLE66CX322P CPU. There the Advanced Crypto Engine (ACE), an arithmetic coprocessor for long integer arithmetic, is used for elliptic curve operations. The coprocessor provides an arithmetic unit that operates with numbers of up to 1120 bit length in long mode and 560 bit in short mode. In principle, the coprocessor always performs elementary operations such as additions, subtractions or shifts on full-length registers. In elliptic curve cryptography, we hardly ever use numbers of more than 512 bit length, so that the overhead for operating with double-length numbers is marginal.

In the Table 2 we compare the run time of Algorithm NINV with the run time of the standard extended Euclidean algorithm for modular inversion as provided by the manufacturer. In both cases the PLL of the CPU runs in an asynchronous mode with maximum possible frequency.

Table 2. Run time of the standard extended Euclidean algorithm and of Algorithm NINV in milliseconds for various bit lengths

Modular inversion algorithm	160 bit	192 bit	256 bit	320 bit
Extended Euclidean	4.80 ms	5.73 ms	7.46 ms	9.16 ms
Algorithm NINV	2.09 ms	2.43 ms	3.16 ms	4.45 ms

The table shows that Algorithm NINV is more than twice as fast as the standard modular inversion algorithm on the SLE66CX322P CPU.

Our implementation of Algorithm NINV chooses an extension factor $f = 3 \cdot 2^k$, for some integer k with $2^k > v$. Then we have $2f - v > 2^{k+2} > f + v$ and $f - v > 2^{k+1}$, so that it suffices to check the bit length of V in steps 2 and 3 of the algorithm. This trick saves some overhead on the coprocessor.

Algorithm NINV does not specify a modular reduction method. Both implementations listed above perform an integer division by using a simple binary subtract-and-correct method. They discard the quotient and keep the remainder. This is adequate here, since the ACE coprocessor can do long integer additions, subtractions and shifts very fast, and the Euclidean quotients are usually quite small, see [8].

We have not implemented any of the optimised gcd algorithms [6, 9, 16] on the SLE66CX322P CPU, since they have been designed for long integers that do not fit into a single CPU register.

In principle, Lehmer's variant [9] of the Euclidean algorithm for long integers can be used to compute the modular inverse in the same way as in Algorithm NINV, since it produces the same quotients as the original Euclidean algorithm, see [8].

6 Conclusion

In practice, many elliptic curve implementations are running on smart card coprocessors that have been designed for RSA cryptography. Taking into account this special design of the Infineon Advanced Crypto Engine (ACE), we have more than doubled the speed of modular inversions used in EC cryptography. This speedup is so significant that it has an observable effect on the speed of some EC algorithms, such as the ECDSA signing procedure.

References

1. I.F. Blake, G. Seroussi, N.P. Smart, *Elliptic Curves in Cryptography,* vol. 265 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 1999.
2. B. Chevallier-Mames, N. Joye, P. Paillier, Faster Double-Size Modular Multiplication from Euclidean Multipliers. *C.D. Walter, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2003*, Springer LNCS vol. 2779, (2003), pp. 214-227.
3. H. Cohen, A. Miyaji, T. Ono, Efficient Elliptic Curve Exponentiation using Mixed Coordinates. *K. Ohta and D. Pei (Eds.): Advances in Cryptology - ASIACRYPT '98*, Springer LNCS vol. 1514 (1998), pp. 51-65.
4. W. Fischer, J.-P. Seifert, Increasing the Bitlength of a Crypto-coprocessor. *B.S. Kaliski Jr, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2002*, Springer LNCS vol. 2523 (2002), pp. 71-81.
5. D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Verlag, 2004.
6. T. Jebelean, A Generalization of the Binary GCD Algorithm. *M. Bronstein (Ed.), 1993 ACM International Symposium on Symbolic and Algebraic Computation, Kiev, Ukraine*, ACM Press (1993), pp. 111-116.
7. M. Joye, P. Paillier, GCD-Free Algorithms for Computing Modular Inverses. *C.D. Walter, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2003*, Springer LNCS vol. 2779, (2003), pp. 243-253.

8. D.E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms,* 3rd ed., Addison-Wesley, 1997.
9. D.H. Lehmer, Euclid's Algorithm for Large Numbers. *American Mathematical Monthly 45*, (1938) pp. 227-233.
10. A.K. Lenstra, E.R. Verheul, Selecting Cryptographic Key Sizes. *J. Cryptology 14 No 4*, (2001) pp. 255-293.
11. R. Lórencz, New Algorithm for Classical Modular Inverse. In *B.S. Kaliski Jr, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2002* Springer LNCS vol. 2523, (2003), pp. 57-70.
12. K. Nakada, *Data Processor and Microcomputer,* US Patent No. 5,961,578, Oct. 5, 1999.
13. P. Paillier, Low-Cost Double-Size Modular Exponentiation or How to Stretch Your Cryptoprocessor. *H. Imai, Y. Zheng (Eds.): Public-Key Cryptography*, Springer LNCS vol. 1560 (1999), pp. 223-234.
14. J.-J. Quisquater, *Encoding System according to the so-called RSA Method, by means of a Microcontroller and Arrangement Implementing this System.* US Patent No. 5,166,978, Nov 24,1992.
15. H. Sedlak, The RSA Cryptography Processor. *Proc. of EUROCRYPT '87*, Springer LNCS vol. 293 (1987), pp. 95-105.
16. J.P. Sorenson, Two Fast GCD Algorithms. *Journal of Algorithms, 16*, (1994), pp. 110-144.
17. J.P. Sorenson, An Analysis of the Generalized Binary GCD Algorithm. `http://euclid.butler.edu/~sorenson/papers/genbin.pdf`.
18. J. Stein, Computational Problems Associated with Racah Algebra. *Journal of Computational Physics 1*, (1967), pp. 397-405.
19. D. Symes, D.J. Seal, *A System for Performing Modular Multiplication.* UK Patent GB 2352309 A, Jan 24, 2001.