A Simpler Sieving Device: Combining ECM and TWIRL

Willi Geiselmann[1], Fabian Januszewski[2], Hubert Köpfer[1], Jan Pelzl[3], and Rainer Steinwandt[4]*

[1] Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik, Am Fasanengarten 5, Universität Karlsruhe, 76128 Karlsruhe, Germany, `geiselma@ira.uka.de`
[2] Mathematisches Institut II, Englerstraße 2, Universität Karlsruhe, 76128 Karlsruhe, Germany, `fabian.januszewski@math.uni-karlsruhe.de`
[3] Horst Görtz Institute for IT-Security, Ruhr University of Bochum, Universitätsstraße 150, 44780 Bochum, Germany, `pelzl@crypto.rub.de`
[4] Department of Mathematical Sciences, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA, `rsteinwa@fau.edu`

Abstract. A main obstacle in manufacturing the TWIRL device for realizing the sieving step of the Number Field Sieve is the sophisticated chip layout. Especially the logic for logging and recovering large prime factors found during sieving adds significantly to the layout complexity. We describe a device building on the Elliptic Curve Method (ECM) that for parameters of interest allows to replace the complete logging part in TWIRL by an off-wafer postprocessing. The postprocessing is done in real time, leaving the total sieving time basically unchanged.
The proposed device is an optimized ECM implementation building on curves chosen to cope with factor sizes as expected in the output of TWIRL. According to our preliminary analysis, for the relation collection step expected for a 1024 bit factorization our design is realizable with current fab technology at very moderate cost. The proposed ECM engine also determines the vast majority of the needed cofactor factorizations. In summary, we think the proposed device to enable a significant decrease of TWIRL's layout complexity and therewith its cost.

Keywords: RSA, NFS, ECM, cryptanalytic hardware

1 Introduction

Lacking paramount theoretical progress in the design of algorithms for factoring integers, in recent years significant efforts have been invested into designing special purpose hardware for factoring. Having in mind a record factorization of a 1024 bit RSA-modulus, at the moment the (General) Number Field Sieve (NFS) seems to be the algorithm of choice, and consequently several proposals for using dedicated hardware to speed up the time-dominating steps of the NFS have been put forward. In particular, for the NFS' linear algebra step significant progress has been achieved [2, 14, 8, 5, 6]—for the most recent designs thinking of a practical implementation for the 1024 bit case does not seem too far-fetched.

* This work was partially supported by a Federal Earmark grant for *Research in Secure Telecommunication Networks* (2005-06).

On the other hand, even the most recent designs that have been proposed for implementing the sieving step of the NFS—the other time-dominating part of the NFS—rely on highly non-trivial technological assumptions: for mesh-based devices along the lines of [2, 7, 9, 10] no practically promising parameter set for the 1024-bit case has been proposed so far, the TWIRL device [23, 15] involves rather large chip sizes along with a non-trivial layout, and for SHARK [4] the actual implementation of the underlying butterfly transport system is technologically challenging. From a practical point of view, finding modifications or alternatives to the existing proposals that are of comparable performance but closer to existing fab technology is highly desirable.

In this contribution we describe a device that allows to remove the complete "diary circuitry" from TWIRL (see [15, Appendix A.7]). This part of TWIRL adds significantly to the layout complexity, but unfortunately seems vital for the recovery of *largish* prime factors found during sieving. Below we show that for relevant parameter choices it is feasible to omit the recording of prime factors during sieving, and to recover them in a postprocessing phase with an optimized implementation of the Elliptic Curve Method (ECM): Our design builds on elliptic curves chosen to cope with factor sizes as expected in the output of TWIRL. Specifically, for the parameters currently considered as realistic for the 1024-bit case, according to our preliminary analysis the proposed ECM engine can be implemented on chips of standard size at very moderate cost. Additionally, the suggested device computes almost all the required cofactor factorizations. As the required computations can be performed in real time, the overall sieving time remains basically unchanged. In summary, we think the proposed ECM engine to allow for a significant decrease of TWIRL's layout complexity and therewith its implementation cost. Due to the very modest hardware requirements our approach may be of independent interest for NFS implementations on "classical" hardware platforms.

Further related work We do not claim the idea of using a dedicated ECM circuitry within the relation collection step of the NFS to be a novel one, e. g., the idea of a parallel ECM engine for smoothness testing is mentioned in [2, 15] already, and [20] proposes a dedicated ECM engine in connection with the SHARK device. The design presented below differs significantly from that in [20] resp. SHARK, however. To the best of our knowledge our proposal is the first that aims at coping with the performance requirements needed to substitute parts of the TWIRL architecture through an ECM engine and has been explored at this level of detail. We deem it an interesting question for future work to what extent the ideas below can also facilitate an implementation of SHARK—and vice versa how to integrate ideas from [20] with TWIRL.

Should we do better? It may look tempting to expand the role taken by ECM in the relation collection, and we considered a hybrid design where an ECM engine replaces the algebraic TWIRL device altogether. However, so far we could not identify a concrete design, where this approach yields a device that is easier to implement than the known proposals.

2 Preliminaries and Parameters

Providing an introduction to the NFS is beyond the scope of this contribution, and we refer to [22] for a survey and to the standard reference [13] for details of the NFS. Section 2.1 gives a brief summary of those aspects of the (relation collection step of the) NFS, that are crucial for our design. Similarly, for an introduction to ECM we refer to [16], but Section 2.2 recalls those aspects that are relevant for describing our design.

2.1 Choice of NFS parameters

In this paper we deal with the NFS' *relation collection* step only. At this we are given two univariate polynomials $f(x), g(x) \in \mathbb{Z}[x]$ sharing a common root m modulo the integer n to be factored:

$$f(m) \equiv g(m) \equiv 0 \pmod{n}.$$

Everything related to $f(x)$ is usually referred to as belonging to the *algebraic side*, and analogously for everything related to $g(x)$ we use the term *rational side*. We are specifically interested in the case of a 1024-bit factorization with $n = 1350 \ldots 7563$ being the number *RSA-1024* as specified in [12]. To this aim we assume the polynomials $f(x)$ and $g(x)$ to be chosen of degree 5 and 1, respectively, as proposed in [23, Appendix B.2] resp. [15, Appendix B]. Both of these specific polynomials are non-monic, and accordingly we define two homogeneous polynomials $N_{\mathbf{a}}(a,b) := |b^5 \cdot f(a/b)|$ (of degree 5) and $N_{\mathbf{r}}(a,b) := |b \cdot g(a/b)|$ (of degree 1). Now the goal of the relation collection step can be phrased as finding pairs of coprime integers (a,b) with $b > 0$ such that both $N_{\mathbf{a}}(a,b)$ and $N_{\mathbf{r}}(a,b)$ split into a product of primes smaller than a smoothness bound $y_{\mathbf{a}}$ resp. $y_{\mathbf{r}}$. At this, we do not require a "full splitting", but allow that $N_{\mathbf{a}}(a,b)$ resp. $N_{\mathbf{r}}(a,b)$ contain up to $\ell_{\mathbf{a}}$ resp. $\ell_{\mathbf{r}}$ "large" prime factors below some semi-smoothness bound $z_{\mathbf{a}}$ resp. $z_{\mathbf{r}}$.

The question of the *concrete* choice of these parameters and of the required *sieving range* for the pairs (a,b) obviously arises. As our device aims at an integration with TWIRL, for the analysis of the 1024-bit case we adopt the parameters from [23] resp. [15, Table 10]:

- On the rational side, the smoothness and semi-smoothness bounds are chosen as $y_{\mathbf{r}} = 3.5 \cdot 10^9$ and $z_{\mathbf{r}} = 4 \cdot 10^{11}$, respectively.
- On the algebraic side, the smoothness and semi-smoothness bounds are chosen as $y_{\mathbf{a}} = 2.6 \cdot 10^{10}$ and $z_{\mathbf{a}} = 6 \cdot 10^{11}$, respectively.
- $2 + 2$ large primes are used, i. e., both on the rational and on the algebraic side we allow for $\ell_{\mathbf{r}} = \ell_{\mathbf{a}} = 2$ large prime factors.

– For the sieving region $-A < a \leq A$, $0 < b \leq B$ we choose $A = 5.5 \cdot 10^{14}$ and $B = 2.7 \cdot 10^8$.

Another figure that is important for analyzing the 1024-bit case in more detail, is the rate at which (a, b)-candidates are output by TWIRL: To be of practical interest, the required test of simultaneous smoothness of $N_{\mathbf{r}}(a, b)$ and $N_{\mathbf{a}}(a, b)$ should be be completed in real-time and not require extensive buffering. Following [23, Appendix A.7], we can expect that a fraction of $\gamma := 2 \cdot 10^{-11}$ of the sieve locations will be output by TWIRL as interesting (a, b)-candidate. With the 1024-bit parameters in [23], TWIRL handles $s_{\mathbf{a}} = 2^{15}$ sieve locations per clock cycle, running at a clock speed of $f := 1$ GHz. If the candidates were output in regular time intervals, we thus had to handle about

$$\varDelta' := f \cdot \gamma \cdot s_{\mathbf{a}} \approx 655$$

sieve locations per second with our ECM engine. At the beginning of the sieving phase more candidates are to be expected, but at this early stage we can simply store the candidates in some buffer, and it seems safe to design our ECM engine to handle $\varDelta := 1000$ candidate (a, b)-pairs/second. The unlikely case of a buffer overflow can be tolerated and is compensated by simply performing slightly more sieving. For each of these candidates we have to determine the norms $N_{\mathbf{r}}(a, b), N_{\mathbf{a}}(a, b)$ and compute the prime factorizations hereof if the smoothness conditions are met. With the mentioned choices for $N_{\mathbf{r}}$, $N_{\mathbf{a}}$, A and B the numbers to be factored can be expected to have no more than 216 bit on the rational and 350 bit on the algebraic side.

It is certainly acceptable to allow the ECM engine to fail for a small fraction of the "good" (a, b)-candidates. Based on software simulations, for the NFS parameters considered here, we decided to use 84 curves on each side, and we estimate that $< 0.5\%$ of the "good" (a, b)-candidates get lost. To further decrease the error, one could implement a version of our design applying more curves and pay for this with an increased chip area. E. g., allowing 128 curves, on the algebraic side, we encountered only 8 integers out of 100, 000 that would be semi-smooth on the algebraic side, but could not be factored by the curves used in our design. The chosen number of 84 curves seems to be an acceptable trade-off between error rate and chip area.

2.2 The Elliptic Curve Method (ECM)

To factor a composite $n \in \mathbb{N}$ with Lenstra's ECM method, one starts by choosing a random point $P = (x : y : 1)$ on a random elliptic curve in Weierstrass normal form. Assuming $\gcd(6, n) = 1$, in affine coordinates that is a solution of a *Weierstrass equation*

$$y^2 = x^3 + ax + b \tag{1}$$

modulo n, where $\gcd(4a^3 - 27b^2, n) = 1$. Then the solutions of (1) modulo a prime divisor q of n constitute a finite abelian group E_q if we adjoin a point

$O = (0 : 1 : 0)$ at infinity (serving as identity element). Lenstra's idea was to apply Pollard's "$p-1$"-method of factorization [21] but to work in E_q instead of $(\mathbb{Z}/q\mathbb{Z})^*$. The key point here is that the order of E_q depends on a, b and q, i. e., not only on q. This allows multiple tries with different curves for the same composite n. Formulae for the group law for curves in Weierstrass form are given in [16]. One can expect that with a certain positive probability—depending on the number of B-smooth integers in $(q+1-2\sqrt{q}; q+1+2\sqrt{q})$—the order $\#E_q$ is B-smooth for a fixed bound B (and $\#E_{q'}$ is not B-smooth for prime divisors $q' \neq q$ of n). Then the computation of

$$\prod_{p<B} p^{e_p} \cdot P, \qquad e_p = \lfloor \log_p(q+1+2\sqrt{q}) \rfloor, \tag{2}$$

yields the prime divisor q of n.

After the computation of (2) we can run a so-called *continuation* as in the case of Pollard's method of factorization. This resembles Shanks' Babystep-Giantstep method. If we choose a second bound $C > B$, a sufficient condition for a continuation of ECM to find q is then that E_q be divisible by a prime p, $B \leq p < C$, and $\#E_q/p$ be B-smooth. We will explain the choice of the continuation in Section 3.2 below.

3 Splitting the Norms in Real Time

The goal of our device is to check whether the candidates (a, b) received (from a diary-free TWIRL) satisfy the rational and algebraic semi-smoothness conditions. If yes, the factorizations of $N_{\mathbf{r}}(a, b)$ and $N_{\mathbf{a}}(a, b)$ are to be computed. These computations are to be done in real time, so that no excessive buffering or a significant increase of the time spent for relation collection becomes necessary. The proposed design naturally splits into two parts which process the received values in a pipelined manner: a *Rational Factorization Unit* and an *Algebraic Factorization Unit.*

3.1 Basic Components

The Rational Factorization Unit is distributed on four identical chips. Each of these chips consists of four parts: The first part, a controller, handles the I/O, distributes the tasks to the other parts on the chip and stores the results. The second part receives the (a, b) pairs from TWIRL through the controller and calculates the rational norm $N_{\mathbf{r}}(a, b)$, where b remains constant during the sieving of one line. After some preprocessing for each line, calculating the rational norm (which can be expected to have no more than 216 bit) therefore reduces to the evaluation of an affine polynomial, i. e., one multiplication and one addition. These operations are performed using a 16 bit adder, some logic for the multiplication and four registers up to a length of 216 bit. The norm $N_{\mathbf{r}}(a, b)$ is forwarded to the trial division pipeline that performs the divisions with all primes/prime powers $\leq 100,000$ and reports the factors found to the controller. The remaining factor of $N_{\mathbf{r}}(a,b)$ (with ≈ 200 bit on average) is then forwarded, through the controller, to the fourth and largest part of the chip, the ECM engine. Details of the trial division pipeline and the ECM engine which factors the remaining part, if the semi-smoothness conditions are met, will be discussed in Section 4.2. If an (a,b)-pair turns out to satisfy the rational semi-smoothness bounds, it is—along with a report encoding the factors found on the rational side—forwarded to the subsequent algebraic factorization unit.

The Algebraic Factorization Unit is realized with five chips and has the same structure as its rational counterpart, but it considers only those (a,b)-pairs where the rational semi-smoothness conditions turned out to be fulfilled already. As on the rational side, first the norm $N_{\mathbf{a}}(a,b)$ has to be computed, which for a constant b amounts to 5 multiplications and 5 additions (using Horner's rule). On the algebraic side we have to deal with larger integers than on the rational side, and we can $N_{\mathbf{a}}(a,b)$ expect to have no more than 350 bit. However only those (a,b)-pairs fulfilling the rational semi-smoothness conditions are forwarded to this device. Therefore on average no more than 80 inputs are expected per chip and second. This reduced number of inputs compensates the larger multiplication/division time. Again, the trial division pipeline and an ECM engine is used to split $N_{\mathbf{a}}(a,b)$ into prime factors.

3.2 Design of the ECM Engine

From a mathematical point of view, our use of ECM on the algebraic and the rational side is the same: We build on an identical set of 84 curves over $\mathbb{Q}$, and for the second phase we use the same improved standard continuation. Due to the different operand sizes, however, the hardware implementation on the algebraic and the rational side is different. In the remaining part of this section we focus on theoretical parameter choices underlying our design. Section 4 discusses aspects related to a hardware implementation.

Choice of Curves Any elliptic curve over a field K with $\operatorname{char} K \nmid 6$ is isomorphic to a curve in Weierstrass form (1). Referring to ideas of Suyama, in [19] Montgomery suggests to use different families of curves to speed up ECM. In fact, it is possible to choose a random elliptic curve E_q over $\mathbb{F}_q$ and to guarantee $d \mid \#E_q$ for any fixed $d \in \{4, 12, 16\}$[5]. Basing on software experiments of one of the authors [11], for our purposes the family proposed by Atkin and Morain in [1] with $d = 16$ seems to perform best.

In [19] Montgomery proposed to use elliptic curves of the form

$$sy^2 = x^3 + tx^2 + x, \qquad \gcd(s(t^2-4), n) = 1. \tag{3}$$

[5] By a celebrated result of Mazur we cannot expect more, because of 16 being the maximal order arising for a torsion subgroup of an elliptic curve over $\mathbb{Q}$.

Equation (3) usually is referred to as *Montgomery form* or *Chudnovsky form*, and [19] gives efficient formulae for the computation on curves of this form. One major advantage is that these formulae enable the evaluation of the product (2) without the use of inversions. Additionally the order of a curve of this type is known a priori to be divisible by 4. Besides being useful for factoring this also implies that not all elliptic curves can be isomorphically transformed into this form. The family of Atkin and Morain may be transformed into Montgomery form. Generation of random curves of this family involves computations on an elliptic curve which cannot be transformed into Montgomery form (due to the reason mentioned above). The aspects of curve generation are discussed in the following paragraph.

Generation of Curves For a composite n we generate a random elliptic curve modulo n and a point on it as follows. According to [1], $S := (12 : 40 : 1)$ is a point of infinite order on $E : Y^2 = X^3 - 8X - 32$ over $\mathbb{Q}$. Therefore we get for every $r \in \mathbb{N}$ a different point $(x : y : 1) := r \cdot S \in E$. According to [1] and [11] every (x, y) yields an elliptic curve E_r in Montgomery form for which we can guarantee $16 \mid \#E_r$ as follows:

Define $\alpha := (x - 9)/(x + y + 16)$ and $\beta := 4\alpha^2 + 2\alpha + 1/2$. Then $\tilde{x} := (\beta + 2\alpha)\beta + 2\alpha + 1/2$, $\tilde{z} := (2\alpha + 1)^2$ yield a point $(\tilde{x} : \tilde{y} : \tilde{z})$ of infinite order on the elliptic curve E_r given by $sY^2 = X^3 + tX^2 + X$. For actual computations we do not need to know the values of $\tilde{y}, \tilde{\alpha}, t$ but the values of

$$\frac{t+2}{4} = \frac{\beta^4}{(2\alpha(2\alpha+1)(4\alpha+1))^2}, \quad \text{and } d = \frac{\tilde{z}^2\beta^2(\beta+2\alpha)}{(2\alpha(4\alpha^2+1))^2(4\alpha^2-1/2)^5}.$$

The first term is needed by the arithmetic described by Montgomery in [19]. The second term d gives a multiple of a square root of the discriminant of the curve. For our design it seems practical to precompute the values of α over $\mathbb{Q}$ for $r \in \{1, \ldots, 84\}$ and to proceed then modulo n to compute $d, \tilde{x}, \tilde{z}$ and $(t + 2)/4$. Nominators and denominators of the coordinates (x, y) grow rather quickly over $\mathbb{Q}$. Since we restricted our setting to the "first" 84 curves of this family the nominators and denominators of the values of α (or 2α) we need to handle are bounded in size by 68 kbits.

Modular reduction of such large numbers takes time. Therefore we partition the set of the values of α as follows. The k-th partition consists of the curves for $r \in \{k, k+21, k+42, k+63\}$. To factor n we then choose randomly[6] $k \in \{1, \ldots, 21\}$ and apply the curves $k, k + 21, k + 42, k + 63$ on n (in that order, because the absolute values of the nominator and denominator of α are smaller for r small). During the computation on an elliptic curve we may precompute the modular reduction needed for the next curve.

[6] In a hardware implementation the needed random value can be determined, e. g., using an LFSR.

First Phase We choose $B = 402$ so that there are 79 primes $p_1, \ldots, p_{79} < B$. Additionally we choose $v = 530$, $e_i := \lfloor \log_p v \rfloor$ and compute $k = \prod_{i=1}^{79} p_i^{e_i}$. Then the first phase consists in computing $Q = k \cdot P$. This can be done efficiently without inversions using Montgomery's formulae [19]. Finally we do a gcd computation to check if this computation already yields a nontrivial divisor d of n (this is necessary for our primality test, see below). If this does yield a nontrivial divisor of n we can continue the computation modulo $n' := n/d$ with $Q' := Q \pmod{n'}$.

We restricted v to the value of 530 instead of $v = (q + 1 + 2\sqrt{q})/16$ corresponding to Equation (2) with $y_{\mathbf{r}} \leq q \leq z_{\mathbf{a}}$ since the probability that a greater power of $p_i^{e_i}$ divides $\mathrm{ord}(P)$ was seen to be very low in simulations for candidates in our setting, see [11]. So the reduction of v leads to a considerable speedup of ECM.

Continuation As second phase for ECM we choose the improved standard continuation as described in [3, Section 3.2] and [11] which may be realized using Montgomery's arithmetic. We choose $C = 9680$ and let $Q = k \cdot P$ denote the result of the first phase. In the continuation we compute sequentially the points $2 \cdot Q, 4 \cdot Q, \ldots, 2t \cdot Q$. Then we can write every prime q with $B \leq q < C$ in the form $q = 2(st \pm r) + 1$, with $1 \leq r \leq t$. This enables us to test if $q \cdot Q = O$ holds modulo a divisor d of n by checking if $2r \cdot Q = \pm(2st + 1) \cdot Q$ holds modulo d. If $l \cdot Q = (X_l : - : Z_l)$ for $l \in \mathbb{N}$ this can be done by checking if $d \mid X_{2r}Z_{2st+1} - X_{2st+1}Z_{2r}$ which in an implementation amounts to computing $\gcd(n, X_{2r}Z_{2st+1} - X_{2st+1}Z_{2r})$. Instead of computing every gcd separately we may compute $d_0 = \gcd(n, T_{0,0})$ where $T_{0,0} = \prod_{r,s} (X_{2r}Z_{2st+1} - X_{2st+1}Z_{2r})$ for all relevant pairs (r, s) which correspond to a prime q (or a pair of primes) as above.

If d_0 is composite this implies that several prime divisors were found at once. In that case we may try to recover those prime divisors by splitting the product $T_{0,0}$ into subproducts $T_{1,0} = \prod_{\text{some } r,s} (X_{2r}Z_{2st+1} - X_{2st+1}Z_{2r})$, $T_{1,1} = T_{0,0}/T_{1,0}$ followed by a computation of $d_1 = \gcd(d_0, T_{1,0})$, $d_2 = \gcd(d_0, T_{1,1}) = d_0/d_1$. If $1 \neq d_1 \neq d_0$ we successfully reconstructed a finer factorization $d_0 = d_1 d_2$. This process may be repeated recursively to eventually compute a decomposition of d_0 into prime factors. We refer subsequently to this structure as the *tree structure* and call the $T_{i,j}$ the *product tree*.

In our design we compute first $T_{1,0}$ and $T_{1,1}$ and then $T_{0,0} = T_{1,0}T_{1,1}$. To have a chance of 50% to split a composite d_0 into nontrivial d_1, d_2 we (once for all) choose the (r, s)-pairs which contribute to $T_{1,0}$ randomly from all pairs. Due to the parameters chosen, there are 1116 primes in the continuation so that there are at most $\lceil \log_2 1116 \rceil = 11$ levels of recursion or equivalently, a product tree of *height* 11, if we want to recover in all cases all information that the continuation may provide. We limit us to a height of 3 so that there are 15 values $T_{0,0}, T_{1,0}, T_{1,1}, \ldots, T_{3,7}$ to be stored. Since we use that tree structure also as a basis for our primality test we compute $d_0 = \gcd(n, T_{0,0})$ and proceed to compute $d_1 = \gcd(d_0, T_{1,0})$ if $d_0 > 1$. Then we compute d_2/d_1 and depending on whether $d_1 > d_2$ or not we continue by computing $d_3 = \gcd(d_1, T_{2,0})$ or $d_3 = \gcd(d_2, T_{2,2})$. Likewise we continue to compute $d_4 = \gcd(d_3, T_{3,i})$ for an $i \in \{0, 2, 4, 6\}$. Consequently we end up with precisely 4 gcd computations after the continuation. See [11] for more details.

We choose $t = 30$ so that there are only 16 different values for r which occur in a representation of a prime q considered in the continuation. Therefore we only need to keep 16 points of the form $2r \cdot Q$ in memory while s runs from 1 to 162. Furthermore a pair (r, s) may represent two primes at once. Our choice of t reduces the number of 1116 primes to precisely 1024 pairs.

Primality Test We know that numbers $n \geq z_{\mathbf{a}}$ are composite and that composite numbers $n < z_{\mathbf{a}}$ are of the form $n = p \cdot q$ with p, q prime, because we assume that the trial division removed all small prime factors less than 100,000. Assuming $p < q$ this leads to $p \leq 774{,}593$. Then for every elliptic curve E_p over $\mathbb{F}_p$ we have $\#E_p \leq 776{,}353$ and we may assume $16 \mid \#E_p$ such that the greatest prime dividing $\#E_p$ is bounded from above by $\tilde{C} := \tilde{v} := 48{,}522$. For the same reason every prime whose square divides $\#E_p$ is less than $\tilde{B} := 220$. We could use ECM with the parameters v, B, C replaced by $\tilde{v}, \tilde{B}, \tilde{C}$ to search for p and thereby checking primality. But the values of v, B, C we choose for factoring do not differ significantly from those given here. In fact, an analysis of the orders of the curves resp. starting points of our family modulo all primes $p \leq 774{,}593$ showed that we may use the same ECM parameters for factoring and primality testing. This enables us to factor and test primality at once. Therefore we need no additional circuitry for a primality test.

There are 1,377,153,921 squarefree composites $n = p \cdot q$ with p, q prime and $100{,}000 \leq p, q \leq 774{,}593$. Table 1 shows the number and percentage of those composites which will not be factored after the given number of partitions of curves. If we assume that every candidate output by the sieving process leads to four composites of the form $p \cdot q$ as discussed above and if we assume that every such composite passes at least 8 curves then Table 1 shows that there are at most $6.6 \cdot 10^7$ semi-smooth candidates which remain composite after the ECM processing. In reality this number should be even smaller.

# curves	4	8	12	16	20	24
missed	$8.4 \cdot 10^6$	74, 000	1, 000	23	1	0.07
%	0.61	$5.4 \cdot 10^{-3}$	$7.6 \cdot 10^{-5}$	$1.7 \cdot 10^{-6}$	$7.4 \cdot 10^{-8}$	$5.1 \cdot 10^{-9}$

Table 1. Number and percentage of squarefree composites $n = p \cdot q$ missed.

Performance of ECM The performance of ECM as a primality test has already been discussed above. Software simulations give evidence that 84 curves suffice to factor most candidates possible. With a set of 84 curves we expect to be able to factor the norms successfully for at least 99.5% of all candidates meeting both the algebraic and the rational semi-smoothness criteria.

4 Hardware Estimates and Implementation Considerations

The objective of our architecture is an AT-efficient design of ECM that for parameters of interest can be implemented with existing technology. In this section we briefly describe the area and time complexity resulting from our choice of algorithms and derive estimates for the overall performance of our design when being applied to the above NFS parameters for RSA-1024.

4.1 Modular Arithmetic

The proposed design requires modular multiplication, squaring, addition, subtraction, and gcd computations. For performing the modular arithmetic operations, we choose Montgomery residues allowing for an efficient method for modular multiplication [18]. Montgomery's algorithm replaces divisions by simple shift operations and, thus, is very efficient in hard- and software. The method is based on a representation of the residue class modulo an integer n. The corresponding n-residue of an integer x is defined as $x' = x \cdot r \bmod n$ where $r = 2^m$ with $2^{m-1} < n < 2^m$ such that $\gcd(r, n) = 1$. Since it is costly to switch always between integers and their n-residue and vice versa, we will perform all computations in the residue class.

Modular subtraction and addition can be computed with a single circuit using simple carry ripple adders (CRA). To guarantee a low latency, operations are done word-wise at an appropriate word size and corresponding number of words. An efficient architecture for modular multiplication is described in [24] and seems suitable for our design. Squaring will be realized with the multiplication circuit since a separate squaring circuit would unnecessarily increase the overall area-time (AT) product. A variant with carry ripple adders has been implemented and analyzed for the use with ECM in [20]. The architecture allows for a word-wise multiplication and is scalable regarding operand size, word size, and pipelining depth.

For the required gcd computations and modular inversions, we adopt the (extended) binary euclidean algorithm [17] which can easily be implemented with the presence of a subtraction hardware and some registers. As additional functionality, two registers must perform a simple shift operation (equivalent to a division by two). Since the (extended) gcd operations are always performed after the actual point operations, we can use internal registers for the computation and do not need additional hardware.

4.2 Factorization Unit

A detailed analysis of the modular arithmetic underlying a hardware implementation of ECM can be found in [20] and its modification for this contribution is summarized in Appendix A.

ECM Cluster: Excluding the time for pre- and post-processing, for the 1024-bit parameters discussed above a single candidate requires a total of 10 ms on the rational and 22 ms on the algebraic side if we assume a (realistic) clock frequency of 240 MHz. For the rational and algebraic ECM unit, the overall transistor count amounts approximately to 290,000 and 340,000, respectively. Assuming standard 0.13 μm CMOS technology, the silicon area required for an ECM cluster, i.e., 4 ECM units together with the reduction unit, is no more than 3.27 mm^2 (rational) and 3.81 mm^2 (algebraic). For details on the complexity estimate of the required area and time, we refer to Appendix A. Figure 1 shows the basic layout of a factorization unit consisting of the norm evaluation, the division pipeline, a central control unit with memory, and the ECM clusters.

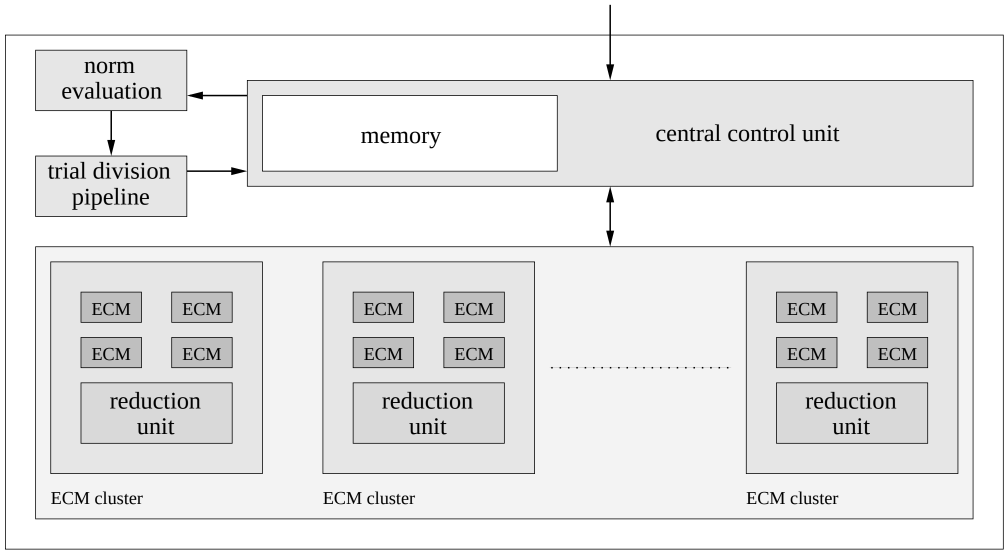

Fig. 1. Basic layout of a factorization unit

Division Pipeline: For the pre-processing of presumably semi-smooth numbers, we perform a trial division by 9592 primes and 108 prime powers up to 100,000 which is realized by a pipelined structure. Once the pipeline is filled it can handle all divisors at a rate of approximately 330 (rational) and 210 (algebraic) numbers per second at a clock rate of 240 MHz. This is sufficient since trial division is done on each chip. The estimated area consumption of the architecture is approx. 60,000 and 75,000 equivalent logic transistors (0.17 mm^2 and 0.21 mm^2) on the rational and algebraic side, respectively.

Central Control Unit: Per chip, we assume a central control unit taking care of all incoming pairs (a, b), the respective factors found during trial division, the computation of the curve parameters, and the corresponding results from the ECM stages. The central control unit can be realized with a standard CPU core attached to some memory for keeping track of the numbers coming from the trial division pipeline and their factorization. We estimate such a unit to consume no more than 1 mm^2 of silicon area.

For a moderate chip size of 147 mm^2, the size of a Pentium 4 processor, we can group 156 rational or 128 algebraic ECM units (39 rational or 32 algebraic clusters) on a single chip.

4.3 Application to TWIRL

For the discussed NFS parameters, when combining our design with TWIRL we estimate that it suffices to handle about 1000 sieve locations per second (cf. Section 2.1). Basing on software simulations, we assume that on the rational side on average 61 curves are used for the factorization of one norm. This requires a computing time of 0.6 seconds per norm. On the algebraic side on average 70 curves are used, this results in a factorization time of 1.5 seconds per algebraic norm. With 4 chips of the size of 147 mm^2 (each including 156 ECM units) up to 1040 rational norms can be factored per second. With 5 chips of the same size (each including 128 ECM units) up to 420 resulting candidates can be checked on the algebraic side. Thus, for the discussed parameters, about 9 chips of standard size should suffice to substitute the diary logic from TWIRL and to compute almost all of the occurring cofactor factorizations.

5 Conclusion

The above discussion shows that the integration of ECM with a diary-free TWIRL results in a sieving design with a significantly simpler layout, but basically the same performance as the original TWIRL. For parameters as currently expected for an NFS-based factorization of RSA-1024, the additional circuitry can be expected to fit on about nine chips of standard size and moderate complexity. Due to its moderate technological requirements, our design might also be of interest for being used in connection with “classical” sieving implementations.

Acknowledgments

We thank Adi Shamir and Eran Tromer for valuable discussions on TWIRL and Thorsten Kleinjung for helpful discussions on the NFS.

References

1. A. Oliver L. Atkin and François Morain. Finding suitable curves for the elliptic curve method of factorization. *Mathematics of Computation*, 60(201):399–405, 1993.
2. Daniel J. Bernstein. Circuits for Integer Factorization: a Proposal. At the time of writing available electronically at `http://cr.yp.to/papers/nfscircuit.pdf`, 2001.
3. Richard P. Brent. Factorization of the tenth and eleventh Fermat Numbers. *Computer Science Laboratory, Australian National Univ., Canberra*, Report TR-CS-96-02:1–25, 1996.
4. Jens Franke, Thorsten Kleinjung, Christof Paar, Jan Pelzl, Christine Priplata, and Colin Stahlke. SHARK: A Realizable Special Hardware Sieving Device for Factoring 1024-Bit Integers. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems; CHES 2005 Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 2005.
5. Willi Geiselmann, Hubert Köpfer, Rainer Steinwandt, and Eran Tromer. Improved Routing-Based Linear Algebra for the Number Field Sieve. In *Proceedings of ITCC '05 – Track on Embedded Cryptographic Systems*. IEEE Computer Society, 2005.
6. Willi Geiselmann, Adi Shamir, Rainer Steinwandt, and Eran Tromer. Scalable Hardware for Sparse Systems of Linear Equations, with Applications to Integer Factorization. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems; CHES 2005 Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2005.
7. Willi Geiselmann and Rainer Steinwandt. A Dedicated Sieving Hardware. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 2003.
8. Willi Geiselmann and Rainer Steinwandt. Hardware for Solving Sparse Systems of Linear Equations over GF(2). In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems; CHES 2003 Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 51–61. Springer, 2003.
9. Willi Geiselmann and Rainer Steinwandt. Yet Another Sieving Device. In Tatsuaki Okamoto, editor, *Topics in Cryptology — CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2004.
10. Tetsuya Izu, Noboru Kunihiro, Kazuo Ohta, and Takeshi Shimoyama. Analysis on the Clockwise Transposition Routing for Dedicated Factoring Devices. In Jooseok Song, Taekyoung Kwon, and Moti Yung, editors, *Information Security Applications: 6th International Workshop, WISA 2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2006.
11. Fabian Januszewski. Ein dedizierter Faktorisierungsalgorithmus auf Basis elliptischer Kurven, 2005. Diplomarbeit, Universität Karlsruhe (Germany), Fakultät für Informatik, Institut für Algorithmen und Kognitive Systeme.
12. RSA Laboratories. The RSA Challenge Numbers. `http://www.rsasecurity.com/rsalabs/node.asp?id=2093`.
13. Arjen K. Lenstra and Jr. Hendrik W. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer, 1993.
14. Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, and Eran Tromer. Analysis of Bernstein's Factorization Circuit. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2002.
15. Arjen K. Lenstra, Eran Tromer, Adi Shamir, Wil Kortsmit, Bruce Dodson, James Hughes, and Paul C. Leyland. Factoring Estimates for a 1024-Bit RSA Modulus. In Chi-Sung Laih, editor, *Advances in Cryptology — ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 55–74. Springer, 2003.
16. Hendrik W. Lenstra. Factoring Integers with Elliptic Curves. *Annals of Mathematics*, 126(2):649–673, 1987.
17. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, USA, 1997.
18. Peter L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.

19. Peter L. Montgomery. Speeding up the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.
20. Jan Pelzl, Martin Šimka, Thorsten Kleinjung, Jens Franke, Christine Priplata, Colin Stahlke, Miloš Drutarovský, Viktor Fischer, and Christof Paar. Area-Time Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method. *IEE Proceedings Information Security*, 152(1):67–78, October 2005.
21. John M. Pollard. A Monte Carlo Method for Factorization. *Nordisk Tidskrift for Informationsbehandlung (BIT)*, 15:331–334, 1975.
22. Carl Pomerance. A Tale of Two Sieves. *Notices of the ACM*, pages 1473–1485, December 1996.
23. Adi Shamir and Eran Tromer. Factoring Large Numbers with the TWIRL Device. In Dan Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2003.
24. Alexandre F. Tenca and Çetin K. Koç. A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm. *IEEE Trans. Comput.*, 52(9):1215–1221, 2003.

A Detailed Cost Estimates

If possible, we provide the exact area and time complexity for the chosen algorithms. However, some algorithms have a non-deterministic runtime and upper bounds on the time complexity are assumed. Hence, the overall estimate should be seen as an upper bound on the time and area complexity. In the sequel, let $m = \lceil \log_2 n \rceil$ be the bitlength of the modulus and let e be the number of words of size w, required to represent a number of size of m bit.

A.1 Complexity of the Basic Building Blocks

For each operation, the area and time complexity is given with respect to the parameters m, e, and p. We will include $T_{init} = 2$ cycles for initialization of the ALU at the beginning of the actual computation.

Memory and Registers We will take the quite realistic area estimates for an 0.13 μm CMOS process from [23]: The area of a single transistor for logic is assumed to be 2.8 μm^2. Due to the highly regular structure, DRAM requires approximately 0.2 μm^2 per bit, equivalent to 0.07 of the area of a transistor for logic[7]. For our implementation, we also require SRAM cells which can be clocked with higher frequencies than DRAM and do not need any refresh cycles. SRAM requires less clock cycles for reading and writing but has the disadvantage of an increased area demand for which we will assume 1.5 gate equivalent (NAND) or 6 transistors.

We assume an Arithmetic Logic Unit (ALU) with a certain amount of internal registers which are used frequently as input and output to the arithmetic functions. For our estimate, we pick SRAM at the cost of 6 transistors per bit for the implementation of such. For tables and larger memory blocks for which we can tolerate a higher latency we choose simple DRAM. For the relatively large

[7] This corresponds to the average area usage of a transistor of a Pentium 4 processor with $55 \cdot 10^6$ transistors on a silicon area of 147 mm^2.

memory blocks, we take care of the additional area for address, row, and column decoding by introducing some overhead.

Modular Addition and Subtraction: With the algorithms and architectures specified in [20], subtraction requires in the worst case $2 \cdot (e+1)$ clock cycles, addition $3 \cdot (e+1)$ clock cycles, where $e = \lceil \frac{m}{w} \rceil$ is the number of words. Hence, the maximum time for an addition or subtraction is bounded by

$$t_{add/sub} = 3 \cdot (e+1) + T_{init}$$

clock cycles. A single full adder (FA) can be built of 3 NAND and 2 XOR gates, summing up to 24 transistors in standard CMOS technology. With w being the the required number of FAs, we can construct the CRA for approximately

$$A_{add/sub} = 24 \cdot w + 500$$

transistors with an assumed overhead of 500 transistors for the internal control logic.

Modular Multiplication: The number of clock cycles per multiplication is given by

$$t_{mul/squ} = \left\lceil \frac{m}{p} \right\rceil \cdot (e+1) + 2p + T_{init},$$

where p is the pipelining depth of the design [24]. For standard CMOS technology, the area consumption of the multiplier is

$$A_{mul/squ} = 4 \cdot (84wp + 25p - 22w)$$

transistors. Note that not all values for p are reasonable (see [24] for possible kernel configurations). A word-width of $w = 64$ bit and a pipelining-depth of $p = 4$ processing elements seems to be a good trade-off for speed and area consumption and is chosen in our context.

(Extended) GCD Computation: The binary gcd architecture for bitlength m requires in the worst case $2m$ subtractions and some cycles for multiplexing (shifting can be hard-wired). For the sake of simplicity, we leave out a detailed analysis of the average runtime of the binary gcd and assume as upper bound

$$t_{gcd} = 2m \cdot ((e+1) + T_{init})$$

clock cycles to finish. Prior each ECM iteration, the precomputation of the required curve parameters needs two modular inversions, which can be computed with the binary extended gcd. The runtime for the algorithm in hardware is at most

$$t_{inv} = (4m+4) \cdot ((e+1) + T_{init})$$

clock cycles [17]. We require no additional register since at the time of the extended gcd computation, all registers for ECM phase 2 are available.

Since we use the ALU for performing the gcd, no additional subtraction circuit or additional registers are required. We assume 2000 transistors for the additional control logic.

Trial Division Pipeline: The trial division by 9592 primes and 108 prime powers up to 100,000 can be realized by a pipelined structure of 10 division circuits. For each division circuit we need two m-bit registers (for the input and the result), two 17-bit registers (for the intermediate result and for the operand), and a 17-bit adder. For the actual division, a simple shift and subtract method is applied, requiring at most m subtractions of 17-bit precision. Since subtractions are performed by additions of the two's complement (plus 1), we can directly use additions by storing the two's complements of the prime powers. The division of a candidate of m bit by a prime power of 17 bit requires no more than $t_{division} = 3 \cdot m + 50$ clock cycles, where the 50 clock cycles are an upper bound on administrative cycles required to initialize the circuit prior computation and to send the result to the central control unit.

Since the division circuit is "too fast" for our purpose, we suggest to use a single circuit for 1000 primes and prime powers by adding required control logic and additional memory to the circuit. Hence, a pipeline of 10 units can handle all divisors at a rate of approximately 330 (rational) and 210 (algebraic) numbers per second at a clock rate of 240 MHz.

The estimated area consumption of the pipeline, including internal control logic, registers, and memory amounts to approx. 60,000 (rational) and 75,000 (algebraic) equivalent logic transistors. Note that the absolute latency of up to 29 ms of the division pipeline is not relevant once the pipeline is filled and a constant throughput is reached.

Reducing the Curve Parameters: The reduction of precomputed parameters for the initialization of ECM requires additional circuitry. We propose to group the ECM units in clusters of four, with a single reduction circuit per cluster. The reduction circuit prepares the next parameters for ECM and operates concurrently to the ECM units. It stores the four (fixed) input values and two intermediate values of size of 68 kbit in DRAM. With a pipelined DRAM to SRAM circuit for prefetching memory blocks and with a simple shift and add circuit based on word-wise addition (word size w), we assume the total area of such a reduction unit to be no more than $A_{reduction} = 15{,}000$ transistors, including memory, required logic for the computation and for the distribution of the results. The required time per reduction is bounded by $t_{reduction} = 1.2 \cdot 10^6$ cycles for the largest input. Since the four inputs are of different size, we can reduce four parameters during the computational phase of ECM.

A.2 Area and Time Complexity of a Single ECM Unit

The precomputation of the curve parameters $d, \tilde{x}, \tilde{z}$ and $(t+2)/4$ can be done with the functionality discussed above, together with some of the registers available from phase 2. The computation amounts to

$$t_{precomp} = 14 \cdot t_{mul/squ} + 7 \cdot t_{add/sub} + 2 \cdot t_{inv} + t_{gcd}$$

clock cycles.

Following [20], phase 1 requires a total of 12 registers. With $t = 30$, the precomputation of the table in phase 2 requires a total of $32 = 2{\cdot}16$ registers of length of m bit for $r \cdot Q$ with $r \in \{1, 4, 6, 7, 9, 10, 12, 15, 16, 19, 21, 22, 24, 25, 27, 30\}$. We need additional 15 registers for building the tree according to Section 3.2, amounting to a total of 59 registers of SRAM, i. e.,

$$A_{SRAM} = 59 \cdot 6 \cdot m$$

equivalent logic transistors. The precomputation time for the table amounts to 7 point duplications and 10 point additions:

$$t_{table} = 7 \cdot t_{point\_dup} + 10 \cdot t_{point\_add} = 95 \cdot (t_{mul/squ} + t_{add/sub})$$

We can determine the runtime and area consumption of both phases on basis of the underlying ring arithmetic and the corresponding upper bound of the runtimes. A setting with $m_r = 216, m_a = 350, w = 64, p = 4, e_r = 4$, and $e_a = 6$ yields $t_{add,r} = 3(e+1) = 17, t_{add,a} = 23$ and $t_{sub,e} = 2(e+1) = 12$, and $t_{sub,a} = 16$ clock cycles. For this configuration, a modular multiplication of full length takes $t_{mul,r} = 280$ and $t_{mul,a} = 626$ clock cycles for the rational bit length $m_r = 216$ and for the algebraic bit length $m_a = 350$, respectively.

For a single gcd we require at most $t_{gcd,r} = 2160, t_{gcd,a} = 4900$ cycles. Hence, the total cycle count for both phases and for a single curves is no more than

$$t_{ECM} = 5600 \cdot t_{add/sub} + 8100 \cdot t_{mul/squ} + 5 \cdot t_{gcd} + t_{precomp}$$

clock cycles including the precomputation time for the table (cf. [11]). Excluding the time for pre- and post-processing, a single candidate on the rational and algebraic side requires a total of $t_{ECM,r} = 2.4 \cdot 10^6$ and $t_{ECM,a} = 5.3 \cdot 10^6$ clock cycles, respectively. If we assume a frequency of 240 MHz, checking a candidate with one curve requires approximately 10 ms on the rational side and 22 ms on the algebraic side. For reading values from the DRAM in phase 2, we assume an efficient SRAM-based prefetch circuit.

For implementing a single ECM unit capable of testing a single curve at a time, we need

$$A_{ECM} = A_{mul/squ} + A_{add/sub} + A_{gcd} + A_{SRAM} + A_{DRAM} + A_k + A_{control}$$

equivalent logic transistors, where $A_{control}$ is the additional logic required to control both phases of ECM, the internal gcd, and fast DRAM to SRAM logic. We assume the control logic to consume no more than $A_{control} = 100{,}000$ equivalent logic transistors which is a fairly conservative estimate[8]. The ECM unit includes a barrel shifter composed of D-flip-flops (48 transistors per bit) for the 589-bit scalar k required for the point multiplication $k \cdot P$ in phase 1. Furthermore, a DRAM memory block stores the (s', r') pairs for all primes between the last prime of the scalar k and $C = 9680$ for phase 2. The pairs (r', s') are sequentially read from memory and are used to control the second phase of ECM. Note that we can compress the information of (r, s) to smaller values (r', s'): The actual value of s is stored in a small 7-bit counter and is simply increased when the 1-bit value s' is equal to '1', leading to a point addition of $2st \cdot Q$ in phase 2. Hence, only a single bit of memory is required for s'. The value of r' points to the corresponding table entry for $r \cdot Q$. With $t = 30$, we require only 4 bit for r'. A total of 5120 bit DRAM for a total of 1024 pairs (resulting from 1116 primes) is required. For the rational and algebraic ECM unit, the overall transistor count amounts approximately to 290,000 and 340,000, respectively. Assuming a standard 0.13 μm CMOS process, a single ECM processor requires no more than 0.81 mm^2 (rational) and 0.95 mm^2 (algebraic) area of silicon.

[8] For comparison: A simple microcontroller such as an 8051 derivate can be realized with 40,000 transistors.