Speeding Up Bipartite Modular Multiplication

Miroslav Knežević, Frederik Vercauteren, and Ingrid Verbauwhede

Katholieke Universiteit Leuven
Department of Electrical Engineering - ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{mknezevi, fvercaut, iverbauw}@esat.kuleuven.be

Abstract. A large set of moduli, for which the speed of bipartite modular multiplication considerably increases, is proposed in this work. By considering state of the art attacks on public-key cryptosystems, we show that the proposed set is safe to use in practice for both elliptic curve cryptography and RSA cryptosystems. We propose a hardware architecture for the modular multiplier that is based on our method. The results show that, concerning the speed, our proposed architecture outperforms the modular multiplier based on standard bipartite modular multiplication. Additionally, our design consumes less area compared to the standard solutions.

Keywords: Bipartite modular multiplication (BMM), Barrett reduction, Montgomery reduction, Public-key cryptography (PKC).

1 Introduction

Public-key cryptography (PKC), a concept introduced by Diffie and Hellman [9] in the mid 70's, has gained its popularity together with the rapid evolution of today's digital communication systems. The best-known public-key cryptosystems are based on factoring *i.e.* RSA [20] and on the discrete logarithm problem in a large prime field (Diffie-Hellman, ElGamal, Schnorr, DSA) [14] or on an elliptic curve (ECC/HECC) [2]. Based on the hardness of the underlying mathematical problem, PKC usually deals with large numbers ranging from a few hundreds to a few thousands of bits in size. Consequently, the efficient implementations of the PKC primitives has always been a challenge.

An efficient implementation of the mentioned cryptosystems highly depends on the efficient implementation of modular arithmetic. Namely, modular multiplication forms the basis of modular exponentiation which is the core operation of the RSA cryptosystem. It is also present in many other cryptographic algorithms including those based on ECC and HECC. In particular, if one uses projective coordinates for ECC/HECC, modular multiplication remains the most time consuming operation for ECC. Hence, an efficient implementation of PKC relies on efficient modular multiplication.

Two algorithms for modular multiplication, namely Barrett [3] and Montgomery [15] algorithms are widely used today. Both algorithms avoid multiple-precision divisions, the operation that is considered to be expensive, especially in hardware. The classical modular multiplication algorithm, based on Barrett's reduction, uses single-precision

multiplications with a precomputed modulus reciprocal instead of expensive divisions [8]. An algorithm that efficiently combines classical and Montgomery multiplications, both in finite fields of characteristic 2, was first proposed by Potgieter [17] in 2002. Published in 2005, a bipartite modular multiplication (BMM) by Kaihara and Takagi [11] extended this approach to the ring of integers.

In this work, we propose a large set of moduli, for which the intermediate quotient evaluation in both Barrett and Montgomery algorithms basically comes for free. Therefore a speed of bipartite modular multiplication, where the Barrett and Montgomery algorithms are the main ingredients, significantly increases. By considering state of the art attacks on public-key cryptosystems, we show that the proposed set is safe to use in practice for both ECC/HECC and RSA cryptosystems. We propose a hardware architecture for the modular multiplier that outperforms the multiplier based on standard BMM method.

The remainder of the paper is structured as follows. Section 2 introduces preliminaries. Section 3 describes the proposed method. In Sect. 4 we give the results of hardware implementations and Sect. 5 discusses the security issues. Section 6 concludes.

2 Preliminaries

In the paper we use the following notations. A multiple-precision n -bit integer A is represented in radix r representation as $A = (A_{n_w-1} \dots A_0)_r$ where $r = 2^w$; n_w represents the number of digits and is equal to $\lceil n/w \rceil$ where w is a *digit-size*; A_i is called a *digit* and $A_i \in [0, r-1]$.

2.1 Classical and Montgomery Modular Multiplication Methods

Given a modulus M and two elements $X, Y \in \mathbb{Z}_M$ where $\mathbb{Z}_M$ is the ring of integers modulo M , the ordinary modular multiplication is defined as:

$$X \times Y \triangleq XY \bmod M .$$

Let the modulus M be an n_w -digit integer, where the radix of each digit is $r = 2^w$. The classical modular multiplication algorithm computes $XY \bmod M$ by interleaving the multiplication and modular reduction phases as it is shown in Algorithm 1. The calculation of the intermediate quotient q_C at step 4 of the algorithm is done by utilizing integer division which is considered as an expensive operation, especially in hardware. The idea of using the precomputed reciprocal of the modulus M and simple shift and multiplication operations instead of division originally comes from Barrett [3]. To explain the basic idea, we rewrite the intermediate quotient q_C as:

$$q_C = \left\lfloor \frac{Z}{M} \right\rfloor = \left\lfloor \frac{\frac{Z}{2^{n+\beta}} 2^{n+\alpha}}{2^{\alpha-\beta}} \right\rfloor \geq \left\lfloor \frac{\left\lfloor \frac{Z}{2^{n+\beta}} \right\rfloor \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor}{2^{\alpha-\beta}} \right\rfloor = \hat{q} .$$

The value $\hat{q}$ represents an estimation of the intermediate quotient q_C . In most of the cryptographic applications, the modulus M is fixed during the many modular multiplications and hence the value $\lfloor 2^{n+\alpha}/M \rfloor$ can be precomputed and reused multiple times.

Algorithm 1. Classical modular multiplication algorithm.

Input: $X = (X_{n_w-1} \dots X_0)_r$, $Y = (Y_{n_w-1} \dots Y_0)_r$, $M = (M_{n_w-1} \dots M_0)_r$ where $0 \leq X, Y < M$, $2^{n-1} \leq M < 2^n$, $r = 2^w$ and $n_w = \lceil n/w \rceil$.

Output: $Z = XY \bmod M$.

```

1:  $Z \leftarrow 0$ 
2: for  $i = n_w - 1$  downto  $0$  do
3:    $Z \leftarrow Zr + XY_i$ 
4:    $q_C \leftarrow \lfloor Z/M \rfloor$ 
5:    $Z \leftarrow Z - q_C M$ 
6: end for
7: Return  $Z$ .
```

Furthermore, an integer division with the power of 2 is a simple shift operation in hardware. Since the value of $\hat{q}$ is an estimated value, some correction steps at the end of the modular multiplication algorithm have to be performed. In his thesis, Dhem [8] determines the values of α and β for which the classical modular multiplication, based on Barrett reduction algorithm, needs at most one subtraction at the end of the algorithm. The improved Barrett algorithm [8], uses the following parameters: $\alpha = w + 3$ and $\beta = -2$.

Montgomery's algorithm [15] is the most commonly utilized modular multiplication algorithm today. In contrast to classical modular multiplication, it utilizes right to left divisions. Given an n_w -digit odd modulus M and an integer $U \in \mathbb{Z}_M$, the image or the Montgomery residue of U is defined as $X = UR \bmod M$ where R , the Montgomery radix, is a constant relatively prime to M . If X and Y are, respectively, the images of U and V , the Montgomery multiplication of these two images is defined as:

$$X * Y \triangleq XYR^{-1} \bmod M.$$

The result is the image of $UV \bmod M$ and needs to be converted back at the end of the process. For the sake of efficient implementation, one usually uses $R = r^{n_w}$ where $r = 2^w$ is the radix of each digit. Similar to the Barrett multiplication, this algorithm uses a precomputed value $M' = -M^{-1} \bmod r = -M_0^{-1} \bmod r$. The algorithm is shown in Algorithm 2.

2.2 Bipartite Modular Multiplication Method

An algorithm that efficiently combines classical and Montgomery multiplications, both in finite fields of characteristic 2, was first proposed by Potgieter [17] in 2002. Extending this approach to the ring of integers, the bipartite modular multiplication (BMM) was introduced by Kaihara and Takagi in [11]. The method efficiently combines a classical modular multiplication method with Montgomery's modular multiplication algorithm. It splits the operand multiplier into two parts that can be processed separately in parallel, increasing the calculation speed. The calculation is performed using Montgomery residues defined by a modulus M and a Montgomery radix R , $R < M$. Next, we outline the main idea of the BMM method.

Algorithm 2. Montgomery modular multiplication algorithm.

Input: $X = (X_{n_w-1} \dots X_0)_r$, $Y = (Y_{n_w-1} \dots Y_0)_r$, $M = (M_{n_w-1} \dots M_0)_r$, $M' = -M_0^{-1} \bmod r$ where $0 \leq X, Y < M$, $2^{n-1} \leq M < 2^n$, $r = 2^w$, $\gcd(M, r)=1$ and $n_w = \lceil n/w \rceil$.

Output: $Z = XYr^{-n_w} \bmod M$.

```

1:  $Z \leftarrow 0$ 
2: for  $i = 0$  to  $n_w - 1$  do
3:    $Z \leftarrow Z + XY_i$ 
4:    $q_M \leftarrow (Z \bmod r)M' \bmod r$ 
5:    $Z \leftarrow (Z + q_MM)/r$ 
6: end for
7: if  $Z \geq M$  then
8:    $Z \leftarrow Z - M$ 
9: end if
10: Return  $Z$ .
```

Let the modulus M be an n_w -digit integer, where the radix of each digit is $r = 2^w$ and let $R = r^k$ where $0 < k < n_w$. Consider the multiplier Y to be split into two parts Y_H and Y_L so that $Y = Y_H R + Y_L$. Then, the Montgomery multiplication modulo M of the integers X and Y can be computed as follows:

$$\begin{aligned}
X * Y &= XYR^{-1} \bmod M \\
&= X(Y_H R + Y_L)R^{-1} \bmod M \\
&= ((XY_H \bmod M) + (XY_L R^{-1} \bmod M)) \bmod M.
\end{aligned}$$

The left term of the last equation, $XY_H \bmod M$, can be calculated using the classical modular multiplication that processes the upper part of the split multiplier Y_H . The right term, $XY_L R^{-1} \bmod M$, can be calculated using the Montgomery algorithm that processes the lower part of the split multiplier Y_L . Both calculations can be performed in parallel. Since the split operands Y_H and Y_L are shorter in length than Y , the calculations $XY_H \bmod M$ and $XY_L R^{-1} \bmod M$ are performed faster than $XYR^{-1} \bmod M$.

2.3 Related Work

Before introducing related work we note here that for the moduli used in all common ECC cryptosystems, the modular reduction can be done much faster than the one proposed by Barrett or Montgomery. Even without any multiplication. This is the reason behind standardizing generalized Mersenne prime moduli (sums/differences of a few powers of 2) [16,1,21].

The idea of speeding up a modular multiplication by simplifying an intermediate quotient was first presented by Quisquater [18] at the rump session of Eurocrypt '90. The method is similar to the one of Barrett except that the modulus is preprocessed before the modular multiplication in such a way that the evaluation of the intermediate

quotient basically comes for free. Preprocessing requires some extra memory and computational time, but the latter is negligible when many modular multiplications are performed using the same modulus.

In [12], Lenstra proposes several ways to generate RSA moduli with any number of predetermined leading (trailing) bits, with the fraction of specified bits only limited by security considerations. He points out that choosing such moduli is beneficial both for storage and computational requirements. Furthermore, Lenstra discusses security issues and concludes that the resulting moduli do not seem to offer less security than regular RSA moduli. Joye [10] enhances the method for generating RSA moduli with a predetermined portion proposed in [12].

3 Speeding Up the Bipartite Modular Multiplication

In both Barrett and Montgomery modular multiplication algorithms, the precomputed values of either modulus reciprocal or modulus inverse are used in order to avoid multiple-precision divisions. However, single-precision multiplications still need to be performed (step 4 of the algorithms above). This especially concerns the hardware implementations, as the multiplication with the precomputed values often occurs within the critical path of the whole design. Section 4 discusses this issue in more detail.

Since the BMM method utilizes both Barrett and Montgomery multiplication algorithms, one needs to precompute both $\mu = \lfloor 2^{n+\alpha}/M \rfloor$ and $M' = -M_0^{-1} \bmod r$. Let us, for now, assume that the precomputed values are both of type 2^γ where $\gamma \in \mathbb{Z}$. By tuning μ and M' to be of this special type, we transform a single-precision multiplication with these values into a simple shift operation in hardware. Therefore, we find a set of moduli for which the precomputed values are both of type 2^γ . A lemma that defines this set is given below:

Lemma 1. *Let $M = 2^n - \Delta 2^w - 1$ be an n -bit positive integer in radix $r = 2^w$ representation with $\Delta \in \mathbb{Z}$, $w \in \mathbb{N}$ and $w < n$. Now, let $\mu = \lfloor 2^{n+\alpha}/M \rfloor$ where $\alpha \in \mathbb{N}$ and $M' = -M_0^{-1} \bmod r$. The following statement holds:*

$$\mu = 2^\alpha \wedge M' = 1 \quad \Rightarrow \quad 0 \leq \Delta \leq \left\lfloor \frac{2^n - 2^\alpha - 1}{2^w(2^\alpha + 1)} \right\rfloor.$$

Proof. To prove the theorem, we first rewrite $2^{n+\alpha}$ as:

$$2^{n+\alpha} = M2^\alpha + \Delta 2^{w+\alpha} + 2^\alpha.$$

Now, the reciprocal μ of the modulus M can be written as:

$$\mu = \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor = 2^\alpha + \left\lfloor \frac{\Delta 2^{w+\alpha} + 2^\alpha}{M} \right\rfloor = 2^\alpha + \left\lfloor \frac{\lambda}{M} \right\rfloor$$

Having that $\mu = 2^\alpha$, the inequality $0 \leq \lambda < M$ must hold. By solving the left part of inequality ($\lambda \geq 0$) we get:

$$\Delta \geq -2^{-w}. \quad (1)$$

Similar, for the right part of inequality ($\lambda < M$) we get:

$$\Delta < \frac{2^n - 2^\alpha - 1}{2^w(2^\alpha + 1)} . \quad (2)$$

From the condition $M' = -M_0^{-1} \bmod r = 1$ it follows that $M \equiv -1 \bmod r$. This is true for all $\Delta \in \mathbb{Z}$. Finally, a condition that the modulus M is an n -bit integer ($2^{n-1} \leq M < 2^n$) makes the last condition for Δ :

$$-2^{-w} < \Delta \leq 2^{n-w-1} - 2^{-w} . \quad (3)$$

Now, from the inequalities (1), (2), (3) and the fact that $\Delta \in \mathbb{Z}$, follows the final condition for Δ :

$$0 \leq \Delta \leq \left\lfloor \frac{2^n - 2^\alpha - 1}{2^w(2^\alpha + 1)} \right\rfloor . \quad \blacksquare$$

The previous theorem defines a set of moduli for which both conditions $\mu = 2^\alpha$ and $M' = 1$ are true. As mentioned earlier, to minimize the number of correction steps in the improved Barrett algorithm [8], we choose $\alpha = w + 3$. Finally, the proposed set is defined as:

$$S : M = 2^n - \Delta 2^w - 1 \text{ where } 0 \leq \Delta \leq \left\lfloor \frac{2^n - 2^{w+3} - 1}{2^w(2^{w+3} + 1)} \right\rfloor .$$

Figure 1 further illustrates the properties of the proposed set. As can be seen, the w least significant bits and the $w + 3$ most significant bits are fixed to be all 1's while the other $n - 2w - 3$ bits can be randomly chosen.

Fig. 1. Binary representation of the proposed set

The evaluation of the intermediate quotient for the improved Barrett algorithm, $\hat{q}$, now becomes equal to:

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{Z}{2^{n+\beta}} \right\rfloor \mu}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{Z}{2^{n+\beta}} \right\rfloor 2^\alpha}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \left\lfloor \frac{Z}{2^{n+\beta}} \right\rfloor 2^\beta \right\rfloor .$$

For $\beta \leq 0$, the previous equation becomes simplified and equivalent to:

$$\hat{q} = \left\lfloor \frac{Z}{2^n} \right\rfloor .$$

Since $M' = 1$, the intermediate quotient for the Montgomery multiplication also gets simplified:

$$q_M = Z \bmod r .$$

Finally, the bipartite modular multiplication for the proposed set of moduli is given in Algorithm 3. After the final addition is performed, one more correction step might be necessary since $0 \leq Z_H + Z_L < 2M$.

Algorithm 3. BMM algorithm for the proposed set of moduli.

Input: $X = (X_{n_w-1} \dots X_0)_r$, $Y = (Y_{n_w-1} \dots Y_0)_r = Y_H r^k + Y_L$, $M = (M_{n_w-1} \dots M_0)_r \in \mathbb{S}$, where $0 \leq X, Y < M$, $r = 2^w$, $0 < k < n_w$ and $n_w = \lceil n/w \rceil$.

Output: $Z = XYr^{-k} \bmod M$.

<pre> 1: $Z_H \leftarrow 0$ 2: for $i = n_w - 1$ downto k do 3: $Z_H \leftarrow Z_H r + XY_i$ 4: $\hat{q} \leftarrow \lfloor Z_H / 2^n \rfloor$ 5: $Z_H \leftarrow Z_H - \hat{q}M$ 6: end for 7: if $Z_H \geq M$ then 8: $Z_H \leftarrow Z_H - M$ 9: end if </pre>	<pre> 1: $Z_L \leftarrow 0$ 2: for $i = 0$ to $k - 1$ do 3: $Z_L \leftarrow Z_L + XY_i$ 4: $q_M \leftarrow Z_L \bmod r$ 5: $Z_L \leftarrow (Z_L + q_M M) / r$ 6: end for 7: if $Z_L \geq M$ then 8: $Z_L \leftarrow Z_L - M$ 9: end if </pre>
---	--

Return $Z \leftarrow Z_H + Z_L$

4 Hardware Implementation

To verify our approach in practice, we implement a set of multipliers that are based on our proposal and compare them with the multipliers that support the original BMM algorithm. Obviously, the mission of the BMM algorithm is to utilize the parallel computation and hence, increase the speed of the modular multiplication. Therefore, in order to compare different designs with the same input size, we define a relative throughput as

$$T_r = \frac{1}{N t_{cp}}$$

where t_{cp} is a critical path delay and N is a number of clock cycles. The total throughput is then obtained as $T = B T_r$, where B is the number of bits processed in $1/T_r$ time.

To maximize the throughput, one obviously needs to decrease both N and t_{cp} . Typically, there are plenty of trade-offs to explore in order to make an optimal (in this case fastest) design. To make an objective comparison, we distinguish between designs that aim at the shortest critical path and the ones that achieve the minimum number of clock cycles. We address each of them separately, in the coming subsections.

4.1 Optimization Goal: Shortest Critical Path

A modular multiplier based on the BMM algorithm, depicted in Fig. 2, consists of four multiple-precision multipliers ($\pi_{H1}, \pi_{H2}, \pi_{L1}, \pi_{L2}$). Apart from the multipliers, the architecture contains some additional adders (Σ_L, Σ_H and Σ). The multiple-precision multipliers are implemented in a digit-serial manner which typically provides a good trade-off between area and speed. The multipliers π_{H1} and π_{H2} assemble together the Barrett modular multiplier that processes the most significant half of Y (that is Y_H). Similarly, the multipliers π_{L1} and π_{L2} form the Montgomery modular multiplier that processes the least significant half of Y (that is Y_L). The results of both multipliers are

finally added together, resulting in $Z = XYr^{-k} \bmod M$. The parameters k and α are chosen such that the execution speed is maximized and the number of correction steps is minimized: $k = \lfloor n_w/2 \rfloor$ and $\alpha = w + 3$.

A choice of the specific architecture is based on the following criteria. The two levels of parallelism are exploited such that the number of clock cycles needed for one modular multiplication is minimized. First, the BMM algorithm itself is constructed such that the Barrett part and the Montgomery part of the multiplier work independently, in parallel. Second, the multiple-precision multipliers π_{H1} and π_{H2} in the Barrett part, and π_{L1} and π_{L2} in the Montgomery part operate with the independent data such that they run in parallel and speed-up the whole multiplication process. The critical path is minimized and consists of one multiplexer, a single-precision multiplier and an adder (bold line, Fig. 2).

Fig. 2. Datapath of the modular multiplier with the shortest critical path based on BMM method

In order to avoid any ambiguity we provide a graph in Fig. 3 which shows the exact timing schedule of separate blocks inside the multiplier. With i ($0 \leq i < k$) we denote the current iteration of the algorithm. Each iteration consists of $n_w + 3$ clock cycles except the first iteration that lasts for $n_w + 1$ cycles.

4.2 Optimization Goal: Minimum Number of Clock Cycles

In order to minimize the number of clock cycles needed for one modular multiplication, the architecture from Fig. 2 is modified as depicted in Fig. 4. Two single-precision multipliers (π_{H3} and π_{L3}), consisting only of a pure combinatorial logic, are added without requiring any clock cycles for calculating their products.

Fig. 3. Timing schedule of the BMM multiplier with the shortest critical path

Fig. 4. Datapath of the modular multiplier with the minimized number of clock cycles based on BMM method

We again provide a graph in Fig. 5 which shows the timing schedule of the multiplier. Each iteration now consists of $n_w + 2$ clock cycles except the first that lasts for $n_w + 1$ cycles.

The critical path of the whole design occurs from the output of the register Z_H to the input of the temporary register in π_{H1} , passing through two single-precision multipliers and one adder (bold line).

4.3 Proposed Multiplier

An architecture of the modular multiplier based on the BMM method with the moduli from the proposed set (see Algorithm 3) is shown in Fig. 6. The most important difference is that there are no multiplications with the precomputed values and hence, the

Fig. 5. Timing schedule of the BMM multiplier with the minimized number of clock cycles

Fig. 6. Datapath of the modular multiplier based on BMM method with the modulus from the proposed set

critical path contains one single-precision multiplier and one adder only. A full timing schedule of the multiplier is given in Fig. 7. The number of cycles remains the same as in the architecture from Fig. 4 while the critical path reduces.

4.4 Results

To show this in practice, we have synthesized 192-bit, 512-bit and 1024-bit multipliers, each with the digit size of 16, 32 and 64 bits. The designs were synthesized using UMC

Fig. 7. Timing schedule of the proposed BMM multiplier

0.13 μm CMOS High-Speed standard cell library with Synopsys Design Vision version C-2009.06-SP3. The results are given in Table 1.

Observing the implementation results, we conclude that our proposed design outperforms the standard BMM design with the shortest critical path for at most 18 %. A design that is based on standard BMM with the minimum number of clock cycles is outperformed by at most 67 %. Furthermore, our design consumes less area than all its counterparts.

5 On the Security of the Proposed Set

In this section we analyze the security implications of choosing primes in the proposed set for use in ECC/HECC and in RSA.

In the current state of the art, the security of ECC/HECC over finite fields $\text{GF}(p^m)$ only depends on the extension degree m of the field [2]. Therefore, the security does not depend on the precise structure of the prime p . This is illustrated by the particular choices for p that have been made in several standards such as SEC [21], NIST [16], ANSI [1]. In particular, the following primes have been proposed: $p_{192} = 2^{192} - 2^{64} - 1$, $p_{224} = 2^{224} - 2^{96} + 1$, $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, and $p_{521} = 2^{521} - 1$. It is easy to verify that for $w \leq 28$ all primes except p_{224} are in S . In conclusion: choosing a prime of prescribed structure has no influence on the security of ECC/HECC.

The case of RSA requires a more detailed analysis than ECC/HECC. First, we assume that the modulus N is chosen from the proposed set. This is a special case of the security analysis given in [12] followed by the conclusion that the resulting moduli do not seem to offer less security than regular RSA moduli.

Next, we assume that the primes p and q that constitute the modulus $N = pq$ are both chosen in the set S . To analyze the security implications of the restricted choice of p and q , we first make a trivial observation. The number of n -bit primes in the set S for $n > 259 + 2w$ is large enough such that the exhaustive listing of these sets is impossible, since a maximum of $2w + 3$ bits are fixed.

Table 1. Synthesis results for the hardware architectures of 192-bit, 512-bit and 1024-bit modular multipliers

Design	n [bit]	w [bit]	Area [kGE]	t_{cp} [ns]	N	T_r [MHz]
Fig. 2	192	16	48.20	2.94	178	1.91
		32	85.66	4.46	52	4.31
		64	212.40	7.29	16	8.57
	512	16	96.31	3.17	1118	0.28
		32	134.10	4.79	302	0.69
		64	259.84	7.49	86	1.55
	1024	16	177.93	3.33	4286	0.07
		32	208.59	5.19	1118	0.17
		64	356.37	7.46	302	0.44
Fig. 4	192	16	50.17	4.34	167	1.38
		32	84.25	6.81	47	3.12
		64	220.73	12.16	14	5.87
	512	16	97.44	4.28	1087	0.21
		32	127.33	6.95	287	0.50
		64	271.54	12.43	79	1.02
	1024	16	169.49	4.48	4223	0.05
		32	198.01	6.91	1087	0.13
		64	341.59	12.22	287	0.29
Fig. 6	192	16	44.65	2.92	167	2.05
		32	83.14	4.18	47	5.09
		64	204.73	7.25	14	9.85
	512	16	95.23	3.10	1087	0.30
		32	137.41	4.37	287	0.80
		64	247.35	7.45	79	1.70
	1024	16	183.01	3.16	4223	0.07
		32	211.07	4.71	1087	0.20
		64	346.40	7.45	287	0.47

The security analysis then corresponds to attacks on RSA with partially known factorization. This problem has been analyzed extensively in the literature and the first results come from Rivest and Shamir [19] in 1985. They describe an algorithm that factors N in polynomial time if $2/3$ of the bits of p or q are known. In 1995, Coppersmith [5] improves this bound to $3/5$.

Today's best attacks all rely on variants of Coppersmith's method published in 1996 [7,6]. A good overview of these algorithms is given in [13]. The best results in this area are as follows. Let N be an n bit number, which is a product of two $n/2$ -bit primes. If half of the bits of either p or q (or both) are known, then N can be factored in polynomial time. If less than half of the bits are known, say $n/4 - \varepsilon$ bits, then the best algorithm simply guesses ε bits and then applies the polynomial time algorithm, leading to a running time exponential in ε . In practice, the values of w (typically $w \leq 64$) and n ($n \geq 1024$) are always chosen such that our proposed moduli remain secure against Coppersmith's factorization algorithm, since at most $2w + 3$ bits of p and q are known.

Finally, we consider a similar approach extended to moduli of the form $N = p^r q$ where p and q have the same bit-size. This extension was proposed by Boneh, Durfee and Howgrave-Graham [4]. Assuming that p and q are of the same bit-size, one needs a $1/(r+1)$ -fraction of the most significant bits of p in order to factor N in polynomial time. In other words, for the case $r = 1$, we need half of the bits, whereas for *e.g.* $r = 2$ we need only a third of the most significant bits of p . These results show that the primes $p, q \in \mathbb{S}$, assembling an RSA modulus of the form $N = p^r q$, should be used with care. This is especially true when r is large. Note that if $r \approx \log p$, the latter factoring method factors N in polynomial time for any general primes $p, q \in \mathbb{N}$.

6 Conclusion

A set of moduli for which the performance of bipartite modular multiplication considerably increases is proposed in this work. The size of the set is determined by the digit-size and the length of the modulus. Since the security level of ECC/HECC does not depend at all on the precise structure of the prime p , our proposed set is safe to be used for constructing underlying fields in elliptic curves cryptography. The case of RSA is also discussed with a conclusion that if used with care ($w \leq 64$ and $n \geq 1024$) our proposed set does not decrease the security of RSA.

Additionally, we propose an architecture for a modular multiplier that is based on our method. The results show that, concerning the speed, our proposed architecture outperforms the modular multiplier based on standard BMM method with no additional area overhead.

Acknowledgment

Work supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State, by FWO project G.0300.07, by the European Commission under contract number ICT-2007-216676 ECRYPT NoE phaseII ,and by K.U.Leuven-BOF (OT/06/40).

References

1. ANSI. ANSI X9.62 The Elliptic Curve Digital Signature Algorithm (ECDSA), <http://www.ansi.org>
2. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press, Boca Raton (2005)
3. Barrett, P.: Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg (1987)
4. Boneh, D., Durfee, G., Howgrave-Graham, N.: Factoring $N = p^r q$ for Large r . In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 326–337. Springer, Heidelberg (1999)
5. Coppersmith, D.: Factoring with a Hint, IBM Research Report RC 19905 (1995)
6. Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 178–189. Springer, Heidelberg (1996)

7. Coppersmith, D.: Small Solutions to Polynomial Equations, and Low Exponent Vulnerabilities. *Journal of Cryptology* 10(4), 233–260 (1996)
8. Dhem, J.-F.: Design of an Efficient Public-Key Cryptographic Library for RISC-based Smart Cards. PhD Thesis (1998)
9. Diffie, W., Hellman, M.E.: New Directions in Cryptography. *IEEE Transactions on Information Theory* 22, 644–654 (1976)
10. Joye, M.: RSA Moduli with a Predetermined Portion: Techniques and Applications. *Information Security Practice and Experience*, 116–130 (2008)
11. Kaihara, M.E., Takagi, N.: Bipartite Modular Multiplication. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 201–210. Springer, Heidelberg (2005)
12. Lenstra, A.: Generating RSA Moduli with a Predetermined Portion. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 1–10. Springer, Heidelberg (1998)
13. May, A.: Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey (2007), <http://www.informatik.tu-darmstadt.de/KP/publications/07/111.pdf>
14. Menezes, A., van Oorschot, P., Vanstone, S.: *Handbook of Applied Cryptography*. CRC Press, Boca Raton (1997)
15. Montgomery, P.: Modular Multiplication without Trial Division. *Mathematics of Computation* 44(170), 519–521 (1985)
16. National Institute of Standards and Technology. FIPS 186-2: Digital Signature Standard (January 2000)
17. Potgieter, M.J., van Dyk, B.J.: Two Hardware Implementations of the Group Operations Necessary for Implementing an Elliptic Curve Cryptosystem over a Characteristic Two Finite Field. In: IEEE AFRICON. 6th Africon Conference in Africa, pp. 187–192 (2002)
18. Quisquater, J.-J.: Encoding System According to the So-Called RSA Method, by Means of a Microcontroller and Arrangement Implementing this System, US Patent #5,166,978 (1992)
19. Rivest, R.L., Shamir, A.: Efficient Factoring Based on Partial Information. In: Pichler, F. (ed.) EUROCRYPT 1985. LNCS, vol. 219, pp. 31–34. Springer, Heidelberg (1986)
20. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* 21(2), 120–126 (1978)
21. Standards for Efficient Cryptography. SEC2: Recommended Elliptic Curve Domain Parameters (2010), <http://www.secg.org>