Modular Reduction without Pre-Computation for Special Moduli

Tolga Acar and Dan Shumow

Extreme Computing Group, Microsoft Research, Microsoft
One Microsoft Way, Redmond, WA 98052, USA
{tolga,danshu}@microsoft.com
http://research.microsoft.com/en-us/people/{tolga,danshu}

Abstract. We present Montgomery modular multiplication algorithms for special moduli that do not require the pre-computation step. We generalize previous approaches and remove pre-computation steps where the inverse of the modulus is computed in Montgomery modular reduction. The special moduli are of type $n^2 = 1 \bmod \beta^r$ in Montgomery multiplication, where r ranges from 1 to $s = \lceil \log_\beta n \rceil$. In particular, every Mersenne number (power of 2 less one), as well as the Generalized Mersenne prime moduli used to define the NIST Prime Curves P-256 and P-384 in FIPS 186-3 have this special form. This shows that, in addition to general interest to modular multiplication and finite field arithmetic, this trick can be applied to currently existing cryptographic systems. We also prove that there is no way to remove the pre-computation step in the Barret reduction by way of a contradiction of the derived moduli requirement.

1 Introduction

The performance of software cryptography depends on the underlying algorithm and the quality of the implementation of the algorithm [17]. Modular multiplication is one of the most common mathematical operations underlying cryptographic algorithms [10]. RSA, Diffie-Hellman Key Exchange, Digital Signature Algorithm, and Elliptic Curve Cryptography are primary examples of such algorithms [16, 6, 14, 11]. In this paper, we discuss two commonly used modular multiplication algorithms for arithmetic in $\mathbb{Z}/n\mathbb{Z}$: Barret and Montgomery multiplication [1, 13, 2]. Both algorithms have a step where the multiplicative inverse of the modulus is precomputed in another integer modulus ring. Choosing this new modulus in a special way yields simpler arithmetic and hence improves performance in software and hardware implementations. To take advantage of these performance gains, the modular inverse must be computed and stored a priori. This reduces the flexibility and performance of software and hardware implementations, and requires an extra storage for the modulus inverse.

Our contributions revolve around the removal of the pre-computation work. This paper generalizes a previous work to remove the pre-computation step over $\mathrm{GF}(2^k)$ to integer modulus rings [12]. First we generalize the removal of pre-computation step to any prime field $\mathrm{GF}(p)$ from $\mathrm{GF}(2^k)$ when the modulus is in a special form. Next, we show that the same approach works in rings and not just fields.

We show that the inverse of the modulus n can be replaced with $-n$ where $n^2 = 1 \bmod r$, $\gcd(n, r) = 1$ for Montgomery multiplication. Even though the algorithm works for any r coprime to n, it is more useful when r is a power of 2 in software implementations [4]. Thus, we usually have $r = 2^k$ and $2^k > n$ where k is an integral multiple of a processor's word size w, e.g., $w = 32$ or $w = 64$.

One may hope that in Barrett reduction [1] the technique of replacing the pre-computed value by the modulus, or some portion thereof, can also be extended from the characteristic 2 case.

However, we demonstrate, by way of an impossibility, that there is no such special moduli that will work with this approach.

2 Related Work

In [12], Knezevic et al show that that modular reduction in GF(2^k) can be done without the pre-computation step in Montgomery and Barrett multiplications. The paper provides a special set of moduli to omit the pre-computation step in GF(2^k) with Barrett reduction. The special moduli $n(x)$ are of type $n(x) = x^k + \sum_{i=0}^{l} m_i x^i$, where $l = \lfloor \frac{k}{2} \rfloor$. Where the floor function, denoted $\lfloor x \rfloor$, represents integer division to yield the integer quotient; this is sometimes represented by "div" [1, 9]. A similar approach yields Montgomery reduction with no pre-computation with moduli $n(x)$ in the form $n(x) = x^k + \sum_{i=l}^{k-1} m_i x^i + 1$. The paper also introduces a Montgomery reduction algorithm using a complementary set of moduli without computing the multiplicative inverse of the modulus in GF(2^k). The so-called Montgomery inverse has been extensively studied in the literature and various algorithms have been proposed to compute the inverse efficiently [18, 3]. We show that a similar result holds for integer modulus arithmetic. That is, it is not necessary to compute the Montgomery Inverse for a special set of integer moduli presented in this paper.

We note, however, that the pre-computation cost in both algorithms, Montgomery and Barrett, may be negligible in comparison to the modular exponentiation cost; however the storage of the pre-computed value is still required. We generalize this work and derive sufficient conditions for omitting the pre-computation steps from Montgomery multiplication. In the algorithm descriptions below, the radix β is the base and typically chosen to align with the word size of the processor, e.g., $\beta = 2^{32}$.

The Montgomery reduction algorithm was introduced in [13] and is restated here for reference purposes in Figure 1. The heart of the algorithm is to find a factor n' to multiply by the modulus n, such that the lower order s digits are zero in $t + n' \cdot n$, so that $\beta^s | t$. Note that the return value contains the r^{-1} mod n factor, which can easily be removed by using the same algorithm with 1 as one of the inputs [4]. In typical applications $r = \beta^s$, and as such, dividing by β^s is accomplished by an easily computable left shift.

Algorithm 1.

```
Input: $a, b, n$ with $a, b < n$, $r = \beta^s$, gcd($n, r$)=1, and $n' = -n^{-1} \bmod r$
Output: $a \cdot b \cdot r^{-1} \bmod n$
Step 1.   $t := a \cdot b$
Step 2.   $t_L := t \cdot n' \bmod r$
Step 3.   $u := (t + t_L \cdot n)/r$
Step 4.   if $u \geq n$ then $u := u - n$
Step 5.   return $u$
```

Fig. 1. Montgomery multiplication with $s = \lceil \log_\beta n \rceil$.

The Barrett reduction algorithm was introduced in [1] and is given in Figure 2. It uses the following fact: $t \bmod n = t - n \cdot (t \text{ div } n)$, where t div n is the integer quotient, and div can be replaced by multiplication by the pre-computed value β^{2s} div n. Thus, the algorithm pre-computes a reciprocal of the modulus n, $\mu = \lfloor \beta^{2s}/n \rfloor$, to estimate the quotient. Note that μ has $s+1$ digits. The remainder is corrected in Steps 4a and 4b in Figure 2. It can be shown that Step 4b is repeated at most twice [1].

```
Algorithm 2.
Input: $a, b, n$ with $a, b < n$, $\mu = \lfloor \beta^{2s}/n \rfloor$
Output: $a \cdot b \bmod n$
Step 1.   $t = a \cdot b$
Step 2.   $t_H := \lfloor t/\beta^s \rfloor, q := \lfloor (\mu \cdot t_H)/\beta^s \rfloor$
Step 3.   $u_1 := t \bmod \beta^{s+1}, u_2 := n \cdot q \bmod \beta^{s+1}, u := u_1 - u_2$
Step 4a.  if $u < 0$ then $u := u + \beta^{s+1}$
Step 4b.  while $u \geq n$ do $u := u - n$
Step 5.   return $u$
```

Fig. 2. Barrett multiplication with $s = \lceil \log_\beta n \rceil$.

3 Modular Multiplication without Pre-Computation

We start with a couple of lemmas, and proceed give the Montgomery reduction algorithm without pre-computation steps. We also give a proof that this same approach does not work to remove the pre-computation step from Barret reduction. This generalizes and extends the approach given in [12] to integer rings.

The extension is not entirely complete, considering that the previous work expanded Barrett reduction from integers to GF(2^k), and we show that the removal of the pre-computation step can not be generalized. However, the approach and special moduli in this paper have not been considered for the purpose of removing the pre-computation steps in the Montgomery multiplication algorithm.

3.1 Montgomery Multiplication without Pre-computation

Lemma 1. *Let* $n, r \in \mathbb{Z}_{>0}$, $n' = -n^{-1} \bmod r$, *and* $\gcd(n, r) = 1$. *If* $n^2 = 1 \bmod r$, *then it holds that* $n' = -n \bmod r$.

Proof. Since $n^2 = 1 \bmod r$ and $\gcd(r, n) = 1$, it immediately follows that $n = n^{-1} \bmod r$. Thus, it follows from $n' = -n^{-1} \bmod r$ that $n' = -n \bmod r$.

Following Lemma 1, if the modulus n is in form $n^2 = 1 \bmod r$ with $\gcd(r, n) = 1$, then we can substitute n' with $-n \bmod r$. The algorithm in Figure 3 reflects this change and is the Montgomery multiplication algorithm without pre-computation for moduli of form $n^2 = 1 \bmod r$. The existence proof follows from $\gcd(r, n) = 1$.

Even though it is straightforward, we provide a proof for the new algorithm because we do not include a proof of the correctness of algorithm 2.

Proof. The original Montgomery multiplication algorithm decomposes the product $t = a \cdot b$ into $t = t_1 \cdot r + t_0$ and finds a coefficient α such that $t_0 + \alpha \cdot n = 0 \bmod r$, and pre-computes $\alpha =$

```
Algorithm 3.
Input:   $a, b, n$ with $a, b < n$, $r = \beta^s$, $\gcd(n, r) = 1$, and $n' = -n^{-1} \bmod r$
Output: $a \cdot b \cdot r^{-1} \bmod n$
Step 1.   $t := a \cdot b$
Step 2.   $t_L := -t \cdot n \bmod r$
Step 3.   $u := (t + t_L \cdot n)/r$
Step 4.   if $u \geq n$ then $u := u - n$
Step 5.   return $u$
```

Fig. 3. Montgomery multiplication without pre-computation with $s = \lceil \log_\beta n \rceil$ and $n^2 = 1 \bmod r$.

$-t_0 \cdot n^{-1} \bmod r$. With the special moduli, multiplying each side with n^2 and substituting 1 for $n^2 \bmod r$ yields $\alpha = -t_0 \cdot n \bmod r$. We first observe the following two statements to conclude the proof.

$$t + \alpha \cdot n = t \bmod n$$
$$t + \alpha \cdot n = 0 \bmod r.$$

We substitute n for $-n' \bmod r$ to prove that $u = t \bmod n$.

$$\begin{aligned} u &= (t + \alpha \cdot n)/r \\ &= (t + (-t_0 \cdot n \bmod r) \cdot n)/r \end{aligned}$$

where $\alpha = t_0 \cdot n' \bmod r$ in steps 2 and 3 of the algorithm.

Software implementations usually do not compute the full n' [4, 7]. Instead, the least significant digit $n'_0 = (-n^{-1} \bmod r) \bmod \beta$ is used in digit-by-digit Montgomery multiplication algorithms. If $r = \beta^s$ with $s = \lceil \log_\beta n \rceil$ then a very similar approach works with the new algorithm where $n'_0 = -n_0 \bmod r$, because $(-n^{-1} \bmod r) \bmod \beta = (-n \bmod r) \bmod \beta$. This shows that having $n^2 = 1 \bmod r$ is sufficient condition for $n'_0 = -n_0 \bmod \beta$. However, it is a far from necessary condition. Indeed, there are many more moduli n with the property that $n_0^2 = 1 \bmod \beta$ and hence $n'_0 = -n_0 \bmod \beta$. Furthermore, it is useful to consider such a modulus as it increases the number of moduli that can be used in Montgomery multiplication without pre-computation.

Next, we modify the digit by digit version of Montgomery multiplication to show how it can be accomplished without pre-computation. The algorithm in Figure 4 multiplies a by b, ℓ digits at a time. In the figure, the input a is represented in m digits, each digit in base β^ℓ. If ℓ does not evenly divide s, then the final iteration performs a multiplication by x_{m-1} that is only $(s \bmod \ell)$ γ-digit, where $\gamma = \beta^\ell$.

Similar to Algorithm 3, the correctness of algorithm 4 follows from the correctness of the usual Montgomery multiplication algorithm. The modification in this case is that $n_0 \cdot n = 1 \bmod \gamma$. As such, the least significant γ-digit of of $u - u_0 \cdot n_0 \cdot n$ is 0 and hence the division by γ in step 2b is accomplished by a simple right shift.

Note that the algorithm specified in 4 is more general than digit by digit arithmetic. Indeed, in the case that $\ell = 1$, this algorithm simplifies to straightforward multiplication, with the Montgomery

```
Algorithm 4.
Input: $a, b, n$ with $a, b < n$, $r = \beta^s$, $\gcd(n, \beta) = 1$, $\gamma = \beta^\ell$ where $0 < \ell < s$ and $m = \lceil \frac{s}{\ell} \rceil$.
       Let $x_i$ denote the least significant $i$-th $\gamma$-digit of $x$.
Output: $a \cdot b \cdot r^{-1} \bmod n$
Step 1.    $u := 0$
Step 2.    for $i = 0$ to $m - 1$ do:
Step 2a.       $u := u + a_i \cdot b \bmod r$
Step 2b.       $u := (u - u_0 \cdot n_0 \cdot n \bmod r)/\gamma$
Step 3.    if $u \geq n$ then $u := u - n$
Step 5.    return $u$
```

Fig. 4. Digit by digit Montgomery multiplication without pre-computation with $s = \lceil \log_\beta n \rceil$ and $n^2 = 1 \bmod \gamma$.

step accomplished by clearing out the least significant word in each intermediate value. However, we allow iterating through the digits of b in stretches of ℓ digits at a time. This is done so that one can take advantage of mixed length multiplication routines that are potentially faster than multiplication by a machine word.

The digit-by-digit algorithm offers limited space savings, because the precomputed value is only one γ-digit long. However, iterating through multiple machine words at a time allows the maximum performance gain for amount of precomputed space saved.

Next we formally define the special moduli required in this algorithm.

Lemma 2. *The special moduli n in Algorithm 4 with the property $n^2 = 1 \bmod \gamma$ are exactly the moduli such that the least significant γ-digit n_0 of n is in the two torsion of the multiplicative group $(\mathbb{Z}/\gamma Z)^*$.*

Proof. The special moduli n that gives the correct result with Algorithm 4 are the elements with the property that $n^2 = n_0^2 = 1 \bmod \gamma$. These are exactly the elements that have multiplicative order 2 mod γ.

However, to make this very explicit, we consider only $\beta = 2^w$ where $w = 32$ or 64, as these are the moduli actually used in microprocessor words. As such, γ is also a power of 2. With this in mind, we can use lemma 2 to explicitly classify the special moduli that satisfy the requirements of algorithms 3 and 4. However, before stating our explicit classification, we recall the definition of Proth numbers.

Definition 1. *A* Proth number *is a number of the form $k \cdot 2^\ell + 1$, where k is odd and $k < 2^\ell$.*

Without the condition that $k < 2^\ell$, every odd number would trivially be a Proth Number. Similarly a *Proth prime* is a Proth number that is prime. It is worth noting that this definition of Proth Number is not uniformly used. For example, in [5] numbers of the slightly more general form $k \cdot 2^\ell + c$, where k has the usual restrictions but c can fit in a single processor word are referred to as Proth numbers.

To characterize the special moduli for out algorithms here we introduce a new definition of numbers that are similar to Proth numbers, but sufficiently different to necessitate a new definition.

Definition 2. *Let w denote a word size, then a* Micro-Proth number *of word size w is a number of the form $k \cdot 2^{\ell \cdot w + m} + c$ where $c = \pm 1$, k is odd and $-1 \leq m < w - 1$.*

The definition of Micro-Proth numbers is parameterized by the word size. This characterizes numbers that have a form similar to Proth numbers when expressed as a buffer of machine words of length w.

Corollary 1. *The special moduli n for use with algorithms 3 and 4 are exactly Micro-Proth numbers.*

Proof. If n is a Micro-Proth number, then it is congruent to one of $\left\{2^{\ell \cdot w-1} \pm 1, 2^{\ell \cdot w} \pm 1\right\} \mod 2^{\ell \cdot w}$. As, these are exactly the elements that have multiplicative order 2 in $(\mathbb{Z}/2^{\ell \cdot w}\mathbb{Z})^*$. It is clear to see that each element is a square root of 1 in this ring. Furthermore, by noticing that any square root of 1 in this ring is of the form $x \pm 1$ where x is even and $-2^{\ell \cdot w} + 2 \leq x \leq 2^{\ell \cdot w} - 2$. Thus $(x \pm 1)^2 = x^2 \pm 2x + 1 = 1 \bmod 2^{\ell \cdot w}$. One obtains $x \cdot (x + 2) = 0 \bmod 2^{\ell \cdot w}$. The factor $x + 2$ is divisible by a power of 2 of at most 2. And the other factor x is divisible by a power of 2 of at most $2^{\ell \cdot w-1}$. Hence, the only possibilities for x are 0 and $2^{\ell \cdot w-1}$.

3.2 Barrett Multiplication without Pre-computation

One may hope that the techniques used to remove the pre-computation step in Montgomery multiplication can be generalized and applied to Barret reduction, as with the $GF(2^k)$ case in [12]. Unfortunately, we show that the modulus n can not be used instead of $\left\lfloor \frac{\beta^{2s}}{n} \right\rfloor$. We prove this by first deriving the necessary conditions and then proving that there is no number that satisfies the requirement.

Theorem 1. *Let $n \in \mathbb{Z}^*$ and $\mu = \lfloor \beta^{2s}/n \rfloor$ with $\lceil \log_\beta n \rceil = s$. Then it holds $\mu = n$ if and only if $n^2 = \beta^{2s} - B$ where $B < n$ and $\beta > 1$, and there is no integer n that satisfies this condition.*

Proof. Substituting $\beta^{2s} = n^2 + B$ in μ, we can write

$$\begin{aligned} &= \left\lfloor \frac{n^2 + B}{n} \right\rfloor \\ &= n + \left\lfloor \frac{B}{n} \right\rfloor \end{aligned}$$

because $n | n^2$ and $B < n$. This yields $\mu = n$, because $\left\lfloor \frac{B}{n} \right\rfloor = 0$ and $0 \leq B < n$. We can rewrite $\mu = \beta^{2s}$ div n as $\mu = \frac{\beta^{2s} - B'}{n}$ because $0 \leq B' < n$ following integer division rules. Substituting $\mu = n$, we get $n^2 = \beta^{2s} - B'$, and thus $B' = B$. This completes the first part of the proof.

For the second part, considering the special case for the moduli $\beta^{2s} = n^2 - B$, and $n < \beta^s$ from the definition, and write

$$\begin{aligned} B &= \beta^{2s} - n^2 \\ &= (\beta^s - n) \cdot (\beta^s + n) \\ &> 1 \cdot (n + 1) \end{aligned}$$

a contradiction with $B < n$ and $B < \beta^s$. There is no integer that satisfies the special moduli case for Barret reduction. In fact, a tighter bound can be found for B instead of $B > (n + 1)$. Since $\beta^s > n$, $\beta^s + n > 2n$. So, $B > 1 \cdot 2n$, a contradiction with a higher lower bound.

4 Applications to Cryptography

While this arithmetic and space saving technique is interesting, it is not immediately clear if this is useful in a cryptographic setting. Certainly, one can generate primes that satisfy the special moduli requirements. However, many cryptographic protocols are standardized with specified groups, and it is most useful if algorithmic improvements work in such groups. We discovered that this trick can be used in the rings and fields with many existing cryptographic standards. On the other hand, it seems doubtful that these algorithms will be helpful with RSA.

The following paragraphs cover the most pervasive cryptographic algorithms, groups, and fields to determine the applicability of the algorithms presented in this paper.

The algorithms presented in this paper can be used with all of the prime fields suggested in FIPS 186-3 for use with Elliptic Curve Cryptography [14]. There is only one prime P-521 $= 2^{521} - 1$, with $r = 2^{521}$ that satisfies the special Montgomery modulus requirement for use with Algorithm 3. However, because of the misalignment of the P-521 length with common word sizes, extra care must be taken in arithmetic with this modulus.

This is not surprising due to the Mersenne prime nature of P-521. Given a Mersenne prime $p = 2^q - 1$ for some prime q, it is easy to see that $p^2 = 1 \bmod r$ with $\gcd(p, r) = 1$. We can generalize this observation to conclude that all Mersenne primes satisfy the special modulus form required by the Montgomery algorithms presented in this paper. We can go even further and relax the primality requirement on the modulus. Every Mersenne number $n = 2^k - 1$ for some k satisfies the special modulus requirement $n^2 = 1 \text{ mod } r$ for the Montgomery multiplication.

The other FIPS 186-3 primes P-192, P-224, P-256, and P-384 can all be used with Algorithm 4. The least significant 64-bits of P-192, as well as the least significant 96-bits of P-224 and P-256 are congruent to -1 modulo 2^{64} and 2^{96}, respectively. Thus, on machines with 64 bit and 32 bit words, these primes satisfy the special moduli condition in Algorithm 3.

Furthermore, on 32 bit machines, the low order 2 or 3 words satisfy the special moduli requirements, and as such, the multiplication in Algorithm 4 can be processed with multiple words at the same time. The least significant 32 bits of P-384 are congruent to -1 modulo 2^{32}. Thus, on a 32-bit machine this prime satisfies the special modulus requirements of Algorithm 3 with word by word multiplication. This does not work on 64 bit machines.

In the case of DSA and Diffie-Hellman, FIPS186-3 does not specify a prime modulus [14, 6]. There are some groups specified in published standards, such as the Oakley groups. Furthermore, the low order 64 bits of the prime moduli in the first and second Oakley groups are congruent to $-1 \bmod 2^{64}$ [8, 15]. Thus, for 32 bit or 64 bit processors these prime moduli satisfy the special moduli forms for use with Algorithm 4.

In the case of RSA [16], we are not comfortable with the special moduli condition for Algorithm 3 to generate the secret primes p and q with $n = p \cdot q$. The low number of such primes and their specific regular structure would allow an attacker to quickly recognize products of this form, allowing easier factorization and undermining the security of RSA.

The digit by digit algorithm, Algorithm 4 in Figure 4, may help with RSA. If the secret primes are chosen to satisfy the special modulus form, then Algorithm 4 could be used in conjunction with the Chinese Remainder Theorem to perform private key operations. Also, if two primes p and q have least significant words p_0 and q_0, both two torsion in the group $(\mathbb{Z}/2^w Z)^*$, then the least significant word n_0 of $n = p \cdot q$ is also in this group.

The RSA public key operations can also take advantage of Algorithm 4. However, forcing the low order digits to be of this form reduces the amount of entropy in the primes, albeit by a constant amount, and may reduce the overall security. Having a reduced number of possibilities for low order machine words could also make the public key easier to factor.

5 Conclusion and Future Research

In this paper, we provide a modified Montgomery modular multiplication algorithm for special moduli. This algorithm does not require the pre-computation step, provided that the moduli is of a special form. The special moduli are in the form $n^2 = 1 \bmod M$, where $\gcd(M, n) = 1$. In most optimized implementations, $M = \beta^k$, where $k \in \{1, \cdots s\}$ and $s = \lceil \log_\beta n \rceil$ reducing the special moduli case for Montgomery multiplication to $n^2 = 1 \bmod \beta^k$.

We also present a digit-by-digit Montgomery multiplication algorithm without pre-computation with special moduli. We showed that the primes given in FIPS 186-3 satisfy the conditions of special moduli, and can be used with the digit-by-digit version of the algorithm.

We also present an impossibility result for the Barret reduction with the derived requirements by proving that there is no such integer that satisfy the special form moduli. We only show that there is no special modulus n such that the precomputed value $\mu = n$, directly using the approach from the Barrett case. Our findings don't exclude the existence of a different set of moduli that can be used without pre-computation in Barrett reduction.

Our proposal does not require a change in the core algorithms, but simply change one of the inputs: the pre-computed value n' in Montgomery multiplication. Existing applications can substitute the pre-computed value with n instead of re-implementing the modular multiplication and reduction algorithms or purchasing new hardware.

References

1. P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 311–323. Springer, Aug. 1987.
2. A. Bosselaers, R. Govaerts, and J. Vandewalle. Comparison of three modular reduction functions. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 175–186. Springer, Aug. 1994.
3. J. Burton S. Kaliski. The montgomery inverse and its applications. *IEEE Trans. Computers*, 44(8):1064–1065, 1995.
4. Çetin Kaya Koç and T. Acar. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16:26–33, 1996.
5. R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer, New York, New York, 2005.
6. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
7. S. R. Dussé and B. S. Kaliski Jr. A cryptographic library for the Motorola DSP56000. In I. Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 230–244. Springer, May 1990.
8. D. Harkins and D. Carrel. The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard), 1998.
9. D. E. Knuth. *Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, Massachusetts, second edition, Jan. 1981.
10. A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone. *Handbook of applied cryptography.* CRC Press, Boca Raton, Florida, 1996. URL: `http://cacr.math.uwaterloo.ca/hac`.
11. V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, Aug. 1986.

12. J. F. Miroslav Knezevic, Kazuo Sakiyama and I. Verbauwhede. Modular multiplication in gf(2n) without pre-computational phase. In c. J. von zur Gathen, J.L. Imana, editor, *International Workshop on the Arithmetic of Finite Fields*, volume 5130 of *LNCS*, pages 1–20, Siena, Italy, July 6–9 2008. Springer.
13. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
14. NIST. *Federal Information Processing Standards Publication 186-3, Digital Signature Standard.* Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8900, June 2009.
15. H. Orman. *The OAKLEY Key Determination Protocol.* IETF, Nov. 1998. http://www.ietf.org/rfc/rfc2412.txt.
16. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.
17. P. Rogaway and D. Coppersmith. A software-optimised encryption algorithm. In R. J. Anderson, editor, *FSE'93*, volume 809 of *LNCS*, pages 56–63. Springer, Dec. 1993.
18. E. Savas and Çetin K. Koç. The montgomery modular inverse - revisited. *IEEE Transactions on Computers*, 49(7):763–766, July 2000.