Precise Bounds for Montgomery Modular Multiplication and Some Potentially Insecure RSA Moduli

Colin D. Walter

Comodo Research Laboratory
10 Hey Street, Bradford, BD7 1DQ, UK
`www.comodo.net`
and
Computation Department, UMIST
PO Box 88, Sackville Street, Manchester, M60 1QD, UK
`www.co.umist.ac.uk`

Abstract. An optimal upper bound for the number of iterations and precise bounds for the output are established for the version of Montgomery Modular Multiplication from which conditional statements have been eliminated. The removal of such statements is done to avoid timing attacks on embedded cryptosystems but it can mean greater execution time. Unfortunately, this inefficiency is close to its maximal for standard RSA key lengths such as 512 or 1024 bits. Certain such keys are then potentially subject to attack using differential power analysis. These keys are identified, but they are rare and the danger is minimal. The improved bounds, however, lead to consequent savings in hardware.

Key words: Cryptography, RSA cryptosystem, Montgomery modular multiplication, differential power analysis, DPA.

1 Introduction

Modular multiplication forms the basis of RSA encryption [11], El-Gamal encryption [5], and Diffie-Hellman key exchange [2] as well as many other cryptographic functions. One algorithm which is particularly efficient for such multiplication is that of Peter Montgomery [9]. Since the algorithm makes modular adjustments unrelated to the magnitude of the accumulating partial product, precise bounds on its output are of interest in order to determine how much space is required for operands and also how many iterations of its main loop need to be performed.

If M is the modulus in question, it is easy to show that the main loop of Montgomery's algorithm will generate an output less than $2M$ if the inputs satisfy that same condition and enough iterations are done [3]. Thus exponentiation can be performed more simply by omitting the final conditional subtraction of M from each multiplication until the last one. However, with standard choices of RSA key length, such as 512 or 1024 bits, $2M$ usually exceeds the word length by just 1 bit. Hence, for efficiency reasons, the conditional subtractions are usually included.

But, recent studies of embedded cryptographic systems have revealed the importance of eliminating conditional statements from code because they may enable timing attacks to be mounted [7]. For example, the conditional subtraction of M occurs with different probability during an exponentiation depending on whether the operation being performed is a square or a multiplication [13]. This can lead to the recovery of the secret key. So, in [12] it was shown that *all* conditional statements, even the very last one, can be removed from exponentiation code using Montgomery modular multiplication if sufficiently many iterations are performed. Hachez and Quisquater [6] subsequently showed that the sufficient conditions given in [12] can lead to an excessive number of iterations. In fact, the problem only arises close to word boundaries.

The main purpose of this paper is to improve further on the efficiency issues which arise from [6] and [12]. Specifically, we establish the precise conditions for avoiding unnecessary iterations, and for selecting optimal hardware for certain common modulus lengths. Unfortunately, inefficiency turns out to be close to its maximum for standard RSA key lengths such as 512 or 1024 bits. For such key lengths the normal $2M$ bound yields a topmost "overflow" digit of either 0 or 1. Because of the different power used during multiplication by zero and non-zero words, this could lead to an attack on the cryptosystem using *differential power analysis* (DPA) [8], [14]. A secondary aim of this paper is therefore to identify the moduli for which this is a risk and assess the danger. The very fortunate conclusion is that almost all standard keys are safe from this form of DPA attack, and for those which are vulnerable the risk is extremely small. As a by-product, we show that hardware savings are possible for all standard RSA moduli by ensuring that the output has the same number of digits as the modulus.

2 Montgomery Modular Multiplication (MMM)

For the notation, suppose the crypto-system is supported by a multiplier that performs k-bit $\times$ k-bit multiplications. Let $r = 2^k$. The $i{+}1$st digit of A in base r will be written $A[i]$ and so the value of A is $A = \sum_{i=0}^{n-1} A[i]r^i$ if it has n digits. Then the appropriate form of the MMM algorithm is as below. It excludes the usual final conditional statement:

```
If P >= M then P <- P-M
```

which would be applied when the inputs A and B are less than M.

Montgomery's Modular Multiplication Algorithm (MMM)

```
{ Pre-conditions: A has n digits base r, each in the range 0 to r−1,
                  and M is prime to r. }
P <- 0 ;
For i <- 0 to n-1 do
Begin
    q <- (-M^-1(P+A[i]B)) mod r ;
    P <- (P + A[i]B + qM) div r ;
    { Invariant: 0 ≤ P < M + B }
End
{ Post-conditions: Pr^n ≡ A×B mod M,  ABr^-n ≤ P < M+ABr^-n }
```

Here M^{-1} denotes the multiplicative inverse modulo r. The computation of q is done with the multiplier using the lowest digits of M, P and B, and provides a result in the range 0 to $r-1$. The choice ensures exact divisibility by r in the second line of the loop. So the MMM procedure computes $A\times B$ mod M with some shift, which is readily seen to be r^n because there are n iterations. Clearly $P < M+B$ is a loop invariant because it holds initially and, assuming it holds at the end of one iteration, it must hold at the end of the next iteration because

$$(P+A[i]B+qM) \text{ div } r \quad < \quad ((M+B)+(r-1)B+(r-1)M)/r \quad = \quad M+B$$

In particular, it must also hold on termination.

To achieve an output bound of $2M$ when A and B are bounded above by $2M$, it may be necessary to perform extra iterations, i.e. to increase n. Thus, if n is big enough that $A[n-1] \le (r-2)/2$ then, with working as above, the bound on the final output becomes $M+B/2$, which is at most $2M$. But, for standard RSA key lengths, many of the inputs A can be expected to have the same number of bits as M, and hence the top non-zero digit of A is often likely to equal or exceed $r/2$. It therefore seems that an additional iteration with a zero digit of A is necessary to achieve the usual output bound of $2M$.

However, a more precise output bound is claimed in the above code for MMM. Thus, let $\alpha_i = \sum_{j=0}^{i-1} A[j]r^{j-i}$. Then, by induction, $\alpha_{i+1} = (\alpha_i+A[i])/r < 1$ for all i. We will now prove that the loop invariant

$$\alpha_{i+1}B \quad \le \quad P \quad < \quad M+\alpha_{i+1}B$$

holds at the end of the iteration involving $A[i]$. Let P_{i+1} denote the value of P at this point and let q_i be the value of q set during the iteration. Prior to the loop commencement, the property $\alpha_0 B = 0 = P_0 < M = M+\alpha_0 B$ holds. This starts the proof by induction. For the inductive step,

$$\begin{aligned}
\alpha_{i+1}B &= (\alpha_i B + A[i]B)/r \\
&\le (P_i + A[i]B + q_i M)/r \\
&= (P_i + A[i]B + q_i M) \text{ div } r \\
&= P_{i+1}
\end{aligned}$$

and

$$\begin{array}{rcl} P_{i+1} & = & (P_i + A[i]B + q_iM) \text{ div } r \\ & = & (P_i + A[i]B + q_iM)/r \\ & < & ((M{+}\alpha_iB) + A[i]B + (r{-}1)M)/r \\ & = & M + \alpha_{i+1}B \end{array}$$

At the end of the last iteration, $\alpha_n = Ar^{-n}$, and so the loop invariant provides:

Theorem 1. *$ABr^{-n} \le P < M{+}ABr^{-n}$ is a post-condition of* MMM.

An alternative derivation of this can be obtained via the partial quotient formed from the successive values of the digit q. Using the same notation, define $\gamma_i = \sum_{j=0}^{i-1} q_j r^{j-i}$. Then it is easy to see that the invariant

$$P_i = \alpha_i \times B + \gamma_i \times M$$

holds at the start of the loop iteration [4]. As in the case of α_i, γ_i satisifes $0 \le \gamma_i < 1$ for all i. Therefore taking $i = n$ for the invariant at the end of the final iteration yields Theorem 1[1].

The theorem improves on $0 \le P < M{+}B$ because $Ar^{-n} < 1$, and it is substantially better if A has zero or non-maximal top digits. It is also the best possible bound in the sense that it places the output in an interval of length M, which is the smallest interval containing a member of each residue class. Indeed, it determines the output P uniquely up to its residue class. Without some knowledge of the residue class of AB and hence of the digits q, this could not be improved.

Notice that the digits of A are processed from least to most significant. So, if it is initially in redundant form, A can be converted on-line to the non-redundant form required. Also, we could represent B and M with radix 2^l rather than 2^k and use a $k{\times}l$-bit multiplier for computing the new value for P. Then the bits of B and M are consumed l bits at a time, propagating carries, and yielding a non-redundant output. Usually dedicated hardware does either that or uses $2k$ full length digit-parallel additions to compute P in a single clock cycle giving a redundant output.

3 Stability

In the context of exponentiation, the output needs to satisfy the same upper bounds as are required for the inputs so that it can be fed straight back in as another input. Under such conditions, the output will remain bounded. Let this bound be $(1{+}\lambda)M$. Then $P < M{+}ABr^{-n} \le (1{+}\lambda)M$ delivers the requirement for stability, namely $M + (1{+}\lambda)^2M^2r^{-n} \le (1{+}\lambda)M$, i.e.

$$(1{+}\lambda)^2Mr^{-n} \quad \le \quad \lambda \tag{1}$$

[1] Incidentally, if M_i is chosen such that $M_iM \equiv 1 \mod r^i$ then $\gamma_ir^i = ((-\alpha_ir^i){\times}B{\times}M_i) \bmod r^i$.

Solving for λ we obtain

$$2^{-1}r^n(1-\sqrt{1-4Mr^{-n}}) \;\leq\; (1+\lambda)M \;\leq\; 2^{-1}r^n(1+\sqrt{1-4Mr^{-n}}) \qquad (2)$$

In particular, this needs to have real roots, i.e. $4M \leq r^n$. But this suffices for the existence of a suitable λ and so for the stability of its use in exponentiation. Hence

Theorem 2. MMM *preserves an input bound* $(1+\lambda)M$ *at output whenever* $4M < r^n$ *and* λ *satisfies* (1)*.*

This theorem therefore determines the minimal number of iterations n which MMM must perform in order to obtain a stable exponentiation process, namely the least value of n which satisfies the given inequality. Thus, assuming an initial input less than M,

- Exponentiation using MMM will converge if MMM consumes at least two more bits of A than there are bits in M.

Now it is clear that if the top of M is not close to a word boundary (i.e. at least 2 bits away from the next one) then the extra iteration with a zero digit of A, which was applied in [12], is unnecessary. Specifically, the condition used there was $2M < r^{n-1}$. That condition was improved in [6] to $M < r^{n-1}$ when $r \geq 4$. The new bound represents an improvement on both: on [12] when $r > 2$, and on [6] when $r > 4$.

Typically $r = 2^8$, 2^{16} or 2^{32}. In the last case, with a standard 512-bit RSA modulus, 17 iterations are performed on 17 digit numbers, whereas with a 510-bit modulus only 16 iterations would be performed on 16-digit numbers. In both cases, the intermediate calculations actually involve 17 digit numbers, but one might still question whether the extra 2 bits in the modulus to achieve a standard key length are worth the probable extra $6\frac{1}{4}\%$ $(= 17/16-1)$ processing time. The most efficient cases of MMM occur when the key length has two fewer bits than a multiple of the word or digit size because an extra iteration is not required.

A standard RSA configuration has a modulus which satisfies $\frac{1}{2}r^{n-1} < M < r^{n-1}$. Prior to concerns about DPA, MMM for this case would have been terminated after $n-1$ iterations and, if the output had $P[n-1] \neq 0$, a subtraction of M would be performed. Consequently, inputs bounded above by r^{n-1} would generate outputs also bounded above by r^{n-1} because of the loop invariant $P < M+B$. We will show that, although the extra iteration is still necessary in order to remove the conditional subtraction and the extra digit is needed for intermediate calculations, at least the extra digit can be avoided in the output.

4 Partial Product Bounds

The extreme values for the I/O bound $(1+\lambda)M$ may be of interest i) for designing hardware and ii) for determining allowable input for the multiplier. Assume M satisfies the inequality of Theorem 2. It is readily verified from (1) that $\lambda = 1$ is always an acceptable value, so that if A and B are bounded by $2M$ then so will be the output P. This viewpoint, using λ, provides a very tight bound on the number of bits required for P in terms of those used in M. In a similar vein, it is easy to check that if A and B are bounded by $\frac{1}{2}r^n$ then the associated value of λ satisfies (1) and so the output P is also bounded by $\frac{1}{2}r^n$. This alternative viewpoint provides bounds on the number of digits required for P. Since the extreme values of λ in (2) coincide for $4M = r^n$ at $\lambda = 1$ where $(1+\lambda)M = \frac{1}{2}r^n$, these values are in some sense the only ones which will work for every acceptable modulus satisfying the condition of Theorem 2.

Intermediate values of P are bounded by $M+\alpha_i B$, and hence by $M+B$ and therefore by $(2+\lambda)M$. So,

Theorem 3. *Suppose the modulus M in* MMM *satisfies $4M < r^n$. If inputs A and B are bounded above by $2M$ (resp. $\frac{1}{2}r^n$), then the output P is similarly bounded above by $2M$ (resp. $\frac{1}{2}r^n$). Moreover, intermediate values of P are bounded above by $\frac{3}{4}r^n$.*

For n satisfying the condition given in the theorem, this determines n as the maximum number of digits needed to represent the various numbers. If n is also the number of digits in M then n is the exact number of digits required for intermediate and final values of P in MMM. Otherwise, the computations require one more digit than M has. So M lies between $\frac{1}{4}r^{n-1}$ and r^{n-1}. This is the case which usually arises for standard RSA key lengths. In the next section we consider *inter alia* whether the extra nth digit is really necessary in such cases. The answer is surprising.

5 Differential Power Analysis Attacks

Cryptosystems may be vulnerable to timing or differential power attacks [8]. Having eliminated any conditional statements from MMM, it is now necessary to consider transition cases where digits might acquire particular values with frequencies which are measurable using power variations. When the value of n has had to be increased because $4M$ has just exceeded a power of r, a large proportion of very small top digit values can be expected to occur in intermediate results and in final products. These might usefully be partitioned into zero and non-zero values as a basis for an attack. Of particular interest are the cases of standard key sizes, such as 512, 768 or 1024 bits when, as usual, they are multiples of the word length k. Since $2M$ is a bound on the output, we find that the top digit is either 0 or 1. This might well provide exactly the handle that an attacker needs to break the cryptosystem.

Using the same method as described in [14], power or EMR traces from a number of digit-by-digit products can be combined from the same long integer multiplication in order to determine whether or not the leading digit of either input is zero. Alternatively, the Hamming weight of the digit might be measured as it travels along the bus to or from memory. The frequencies of zero and non-zero top digits might then be obtained for each long integer multiplication in an exponentiation. The theory presented by Walter & Thompson in [13] shows that the frequency of a zero top digit will be different for squares and multiplications. (Integrating over the probability density functions for the values of the inputs provides a coefficient $\frac{1}{2}\times\frac{1}{2} = \frac{1}{4}$ in the case of a multiplication, and a coefficient $\frac{1}{3}$ for a square.) This enables squares and multiplications to be distinguished and hence secret key bits to be read from the frequency chart if a sufficiently naïve implementation of exponentiation is used, even if Rivest's method is applied to blinding the input text [1, 10]. However, this attack depends on non-zero top digits appearing with sufficient frequency for this difference to be easily measured and for the result to be very reliable for each decision.

Software was constructed for MMM in order to check the theory and gauge the quantity of data that might leak if very small top digit values could be detected. In particular, the frequency of the top digit being non-zero was of interest when M satisfied the most common situation, namely $M < r^{n-1} < 2M$. This revealed that the top digit does not behave as if the distribution of outputs were fairly uniform in the range $[0..2M]$, as one might have expected from the usual $2M$ bound on I/O. Indeed, we will now show that

- in such standard circumstances the top output digit from MMM is non-zero only very exceptionally.

In fact, the final iteration, which was only just necessary, performs in effect an extra shift down of the AB term in the upper bound $M{+}ABr^{-n}$, making it much smaller than the M component. Thus, from the property $P < M{+}ABr^{-n} < M{+}4M^2r^{-n}$ of inputs less than $2M$, one can deduce that the output satisfies $P < r^{n-1}$ if $M{+}4M^2r^{-n} < r^{n-1}$. Using the Taylor series expansion for $(1{+}16r^{-1})^{1/2}$ shows that this condition is satisfied when $M < r^{n-1}(1{-}4r^{-1})$. So,

Theorem 4. *For inputs less than $2M$ and n satisfying the condition of Theorem 2, the property $M < r^{n-1}(1{-}4r^{-1})$ guarantees that the output of* MMM *satisfies $P < r^{n-1}$.*

Of course, for M close to this upper bound, outputs re-used as inputs satisfy a better bound than $2M$. So it makes sense to see under what conditions $(n{-}1)$-digit inputs will provide $(n{-}1)$-digit outputs. In this case, with similar working to that just given, the condition that needs to be satisfied is $M{+}r^{n-2} < r^{n-1}$. Hence,

Theorem 5. *For inputs of $n{-}1$ digits and M satisfying the property $M < r^{n-1}(1{-}r^{-1})$ the output of* MMM *also has at most $n{-}1$ digits.*

For standard configurations using recommended RSA key lengths and typical off-the-shelf multipliers, n satisfies $\frac{1}{2}r^{n-1} < M < r^{n-1}$, giving the most significant non-zero digit of M in the interval $[\frac{1}{2}r, r{-}1]$. So the condition in Theorem 5 will hold almost always because the exception requires the top digit of M to be $r{-}1$ and the probability of this is only 1 in 2^{k-1}. This is certainly unlikely for 16- or 32- bit digits, though occasionally the case if only 8-bit words are employed.

Hence, under standard conditions, a DPA attack which attacks "overflow" output digits in the way described above is unlikely to succeed because the top non-zero digit of M will usually have an unsuitable value. Of course, the same cannot be claimed if M occupies n digits and its topmost digit is just 1. As in the standard case, $2M$ is a poor upper bound for the inputs and outputs during an RSA exponentiation. The width of the interval in inequality (2) is still reasonably large, enabling λ to be chosen fairly small. This will again provide an upper bound only marginally larger than M and average output values close to $M/2$ so that in the region of half the outputs will be expected to have a non-zero top digit. Hence the DPA attack might become feasible.

Between these two situations, consider the exceptional case with $r^{n-1}(1-r^{-1}) < M < r^{n-1}$. This is, in fact, the only case for which the inputs and outputs will have a different number of digits from the modulus M. By taking the maximum lower bound for $(1+\lambda)M$ given in inequality (1), namely when $M = r^{n-1}$, we know that $2^{-1}r^n(1-\sqrt{1-4r^{-1}})$ is an upper bound which will hold for the output if it holds for the inputs. Using the Taylor series expansion for $\sqrt{1-x}$, we obtain $r^{n-1}+r^{n-2}+2r^{n-3}+5r^{n-4}+14r^{n-5}+...$ as a suitable bound on inputs which is preserved. Since the residues $\bmod M$ are expected to be uniformly distributed, we can expect at most $r^{-1}+2r^{-2}+5r^{-3}+14r^{-4}+...$ of the outputs to be greater than r^{n-1}, i.e. to have a non-zero nth digit. Probably less than half this number will occur because such outputs duplicate residues $\bmod M$ which are less than r^{n-1}.

So, for an 8-bit multiplier, $r = 256$ would enable such non-zero digits to be detected with sufficient frequency for squares and multiplications to be distinguished only if at least several thousand exponentiations were performed. Even then, for a given multiplicative operation in the exponentiation scheme, only several tens of top digits will be non-zero over the whole sample. The natural variance will mean that the frequency of these non-zero digits will not be sufficiently stable to distinguish reliably between squares and multiplies in the exponentiation. Hence it will be very hard to determine the bits of the secret exponent unless other data is available. For a 12-bit or larger multiplier, a standard lifetime bound of 10k exponentiations would certainly make such distinctions impossible. In such cases, one could afford to re-introduce a conditional subtraction to keep the result of modular multiplication less than r^{n-1}:

```
If M[n-1] = 0 and P[n-1] ≠ 0 then P <- P-M
```

Although this leads to rare timing variations, little is lost if power or EMR variations reveal these cases anyway. Of course, if the modulus is known to the attacker and is seen to be one of these exceptional cases, and the input text is not blinded and individual conditional subtractions can be observed, then the attacker simply has to simulate the exponentiation, determine whether a square or a multiplication would generate an observed subtraction and progressively reconstruct the exponent from this information. Hence, as a matter of course, the input should be blinded by, for example, multiplying it by a random number before the exponentiation and performing an appropriate division afterwards [10].

One can conclude that, in general, key lengths which are multiples of the multiplier word length are close to the least efficient as far as Montgomery multiplication is concerned, but that they are usually no less safe. Potentially unsafe moduli have been identified, but any lack of security should only affect poorly designed implementations with very small multipliers, such as 8-bit multipliers. If these exceptional moduli are excluded or an appropriate conditional subtraction included for them, then hardware savings can be made by not implementing the digit of index n for the output register.

6 Exponentiation

A consequence of the extra, unwanted factor r^{-n} in the MMM output is that every exponentiation has to perform pre- and post- processing. A pre-computed value for r^{2n} mod M is Montgomery-multiplied with the initial text T requiring exponentiation. This introduces an extra factor of r^n mod M to the result of every subsequent multiplication compared to the result obtained if classical modular multiplication were performed instead. To remove this extra factor, a final Montgomery multiplication by 1 is performed. In [12] it was observed that taking $A = 1$ in the version of MMM there produced a result less than M, which therefore needed no further conditional subtraction.

Here, taking $A = 1$ gives a bound $P < M+Br^{-n}$ which, when combined with $B < 2M < \frac{1}{2}r^n$, yields $P \leq M$, as before, because P is an integer. Of course, $P = M$ means the initial text T must have satisfied $T \equiv 0 \bmod M$. In the context of cryptography, T must be less than M. Undoubtedly it should not equal 0, but if it were, then every intermediate result would also be 0, giving 0 as the final output. So here also, with the better value for n than [12] or [6],

Theorem 6. *For inputs less than* $2M$ *and* n *satisfying the condition of Theorem 2, a conditional final subtraction to obtain a result in the interval* $[0, M-1]$ *is unnecessary in exponentiation using the above version of the* MMM *algorithm with the usual pre- and post- processing.*

7 Conclusion

By obtaining the best possible bounds on the output, we have shown exactly how many iterations are necessary in an implementation of Montgomery's modular multiplication algorithm in order to avoid the conditional statements which may be subject to timing attacks in cryptographic hardware. This includes eliminating the final conditional subtraction from an exponentiation in which the usual pre- and post- processing of the Montgomery constant r^n mod M occurs.

The investigation suggested that it might be both cryptanalytically unsafe and inefficient to use certain moduli which have standard key lengths, as these are normally equal to a multiple of the multiplier word length. However, the potentially weak moduli have been identified as a very small set and the weakness shown to be very slight: the security of these weaker moduli is only in question from DPA attacks where a small multiplier is used in an implementation without input blinding.

For standard RSA configurations, the conditional statements are avoided by a single extra iteration of the multiplication algorithm. If the weaker moduli can be avoided completely then the output has the same number of digits as the modulus and some hardware savings can be made. But, if the weaker moduli cannot be avoided then, in combination with input blinding, a conditional statement can be safely re-introduced to keep the output to the same number of digits as the modulus so that the hardware savings can still be made.

References

1. D. Chaum, *Blind Signatures for Untraceable Payments*, Advances in Cryptology – Crypto '82, R. L. Rivest, A. T. Sherman & D. Chaum (editors), Plenum Press, New York, 1982, 199–203
2. W. Diffie & M. E. Hellman, *New Directions in Cryptography*, IEEE Trans. Info. Theory, **IT-22**, no. 6 (1976), 644–654
3. S. E. Eldridge, *A Faster Modular Multiplication Algorithm*, Intern. J. Computer Math., **40** (1991), 63–68
4. S. E. Eldridge & C. D. Walter, *Hardware Implementation of Montgomery's Modular Multiplication Algorithm*, IEEE Trans. Comp. **42** (1993), 693-699
5. T. El-Gamal, *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. Info. Theory, **IT-31**, no. 4 (1985), 469–472
6. G. Hachez & J.-J. Quisquater, *Montgomery exponentiation with no final subtractions: improved results*, Cryptographic Hardware and Embedded Systems (Proc CHES 2000), C. Paar & Ç. Koç (editors), Lecture Notes in Computer Science, **1965**, Springer-Verlag, 2000, 293–301
7. P. Kocher, *Timing attack on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology – Crypto '96, N. Koblitz (editor), Lecture Notes in Computer Science, **1109**, Springer-Verlag, 1996, 104–113
8. P. Kocher, J. Jaffe & B. Jun, *Differential Power Analysis*, Advances in Cryptology – Crypto '99, M. Wiener (editor), Lecture Notes in Computer Science, **1666**, Springer-Verlag, 1999, 388–397
9. P. L. Montgomery, *Modular multiplication without trial division*, Mathematics of Computation, **44** (1985), no. 170, 519–521
10. R. L. Rivest, *Timing cryptanalysis of RSA, DH, DSS*, Communication to sci.crypt Newsgroup, 11 Dec 1995
11. R. L. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, **21** (1978), 120–126
12. C. D. Walter, *Montgomery Exponentiation Needs No Final Subtractions*, Electronics Letters, **35**, no. 21, October 1999, 1831–1832
13. C. D. Walter & S. Thompson, *Distinguishing Exponent Digits by Observing Modular Subtractions*, Topics in Cryptology – CT-RSA 2001, D. Naccache (editor), Lecture Notes in Computer Science, **2020**, Springer-Verlag, 2001, 192–207
14. C. D. Walter, *Sliding Windows succumbs to Big Mac Attack*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 286–299