Is there Safety in Numbers against Side Channel Leakage?

Colin D. Walter

Computation Department, UMIST
PO Box 88, Sackville Street, Manchester, M60 1QD, UK
www.co.umist.ac.uk

Abstract. The consequences of two recent attacks on RSA implementations are investigated. Each suggests the potential of increased weakness in the embedded cryptosystem when key length is increased. One does not realise this potential, but the other gives every indication of doing so. In that case, when a single k-bit multiplier is used, increasing the key length provides more data per key bit to attack. Consequently, although the mathematical strength may be improved, the implementation becomes less safe as key length is increased.

1 Introduction

So-called *side channel attacks* on smartcards to discover secret keys contained therein follow a well-established tradition pursued by the military and secret services, and exemplified by the long-running Tempest project of the US [24]. That project concentrated on detecting and obscuring electro-magnetic radiation (EMR) and led to both heavily shielded monitors (those based on tubes) and TV detector vans. EMR can be, and is, used to break smartcards – but with a somewhat smaller aerial, one some 3mm long or less [5]. If correctly placed and set up with sufficiently sensitive equipment, these can detect useful variations in EMR non-invasively.

Side-channel leakage occurs through data dependent variation in the use of resources such as time and hardware. The former results from branching in the code or compiler optimisation [9, 2], and the latter manifests itself through current variation as well as EMR [10, 13, 12]. For the RSA crypto-system [16], conditional modular subtractions should be removed to make the time constant [18, 17, 20]. Bus activity is the major cause of power variation, with a strong relationship between it and the Hamming weight of the data on the bus. Instructions and memory locations pass along the bus and, in the context of the limited computational resources of a smartcard, potentially also large quantities of data. This is partly solved by encryption of the bus [1, 11].

In all the popular crypto-systems used in practice where the key length is variable, the greater the key length, the greater the *mathematical* strength of the system against attack is believed to be. Indeed, a brute force attack will take time exponential in the key length. However, longer key lengths require more

computation for encryption and decryption. Hence there is more data which leaks through timing, power and EMR variation. In an embedded crypto-system to which an attacker has access, such as a smartcard, a valid question to ask is whether or not the increased data from side channel leakage actually makes longer keys *more* vulnerable to attack?

In the symmetric crypto-systems of DES, 3-DES and AES [22, 23], the block length is fixed and the number of rounds is proportional to the key length (comparing DES with 3-DES, and AES with different choices for Nk). Hence the data leakage is also proportional to the key length and the *implementation* strength of the cipher is unlikely to decrease as key length increases.

However, public key cryptography such as in RSA, DSA, ECC, Diffie-Hellman or El-Gamal [16, 21, 14, 3, 4], usually involves exponentiation in some form, where the block length and exponent are proportional to the key length. Assuming multiplication of double-length arguments takes four times the time for single-length arguments on the same hardware, time is proportional to the cube of the key length. Consequently, *more* leaked data is available per key bit as key length grows. Indeed, if the multiplicative operations of the exponentiation are performed sequentially using one of the standard algorithms [7, 8] and no form of blinding, then there is more data per exponent bit for longer key lengths and one should expect the implementation strength to decrease.

Two recent attacks do *not* appear to become more difficult as key length is increased. Here we look at these in more detail: [20] is a timing attack which is apparently independent of key length, whilst [19] is a differential power analysis (DPA) attack in which key bits are determined independently with each using all available data. In the former case, the attack is does, in fact, become more difficult for larger keys. However, in the latter case the data available for deciding each secret key bit is proportional to the cube of the key length, so that the attack becomes easier with longer key lengths.

2 Security Model

The contexts for the two attacks [19, 20] are slightly different, but, for convenience, in both cases we assume a similar scenario. In both cases, a smartcard is performing RSA with limited resources and must be re-usable after the attack. So the attacker is very limited in what he is allowed to do: he can only monitor side channel leakage. He cannot choose any inputs, nor can he read inputs or outputs, and he does not even know the "public" part of the key being used. However, he is allowed to know the algorithms involved, perhaps as a result of previous destructive studies of identical cards, insider information and publicity material. His goal is to determine the secret exponent D, from which we assume he can obtain the (public) modulus M, whether or not the Chinese Remainder Theorem has been used.

For the timing attack [20] we assume the attacker can observe occurrences of conditional subtractions of the modulus at the end of each long integer multiplicative operation with greater than evens chance of being correct. We also

assume that the same exponent is used for all exponentiations which he observes. For the DPA attack [19] we assume the attacker can observe power or EMR variations which bear some connection with the Hamming weights of the inputs to a single multiplier. We also assume that the exponent may have been masked by the addition of a (say 32-bit) random multiple of $\phi(M)$ [9]. In both cases, any further assumptions about the hardware and the various algorithms used to perform the exponentiation will be given later, as necessary.

3 Notation

As above, we assume an n-bit modulus M and private exponent D for the RSA crypto-system. Ciphertext C has to be converted to plaintext $C^D \bmod M$ using a small k-bit multiplier. Hence, except for the exponent, the n-bit numbers X involved in the exponentiation are represented using base $r = 2^k$ and (non-redundant) digits $x_i (0 \le i < s)$ in the range $[0, r)$. Thus $X = \sum_{i=0}^{s-1} x_i r^i$.

The exponent D is represented with a different base, typically 2 or 4, depending on the exponentiation algorithm. Exponentiation is usually performed using the square-and-multiply algorithm, processing the exponent bits in either order, or the generalisation of the most-to-least significant case, called m-ary exponentiation [7, 8], in which D is represented in radix m using, say, t digits, and some powers of $C^{(i)} = C^i \bmod M$ $(1 \le i < m)$ which are pre-computed:

The m-ary (Modular) Exponentiation Algorithm

```
C(1) := C ;
For i := 2 to m-1 do
     C(i) := C(i-1)×C mod M ;
P := C(d_{t-1}) ;
For i := t-2 downto 0 do
Begin
     P := P^m mod M ;
     If d_i ≠ 0 then P := P×C(d_i) mod M ;
End ;
```

Output: $P = C^D \bmod M$

The modular products here are too large for the smartcard multiplier to perform in one operation. Typically a form of Montgomery's modular multiplication algorithm is used [15]. This gives an output R related to $A \times B \bmod M$ via a power of r scaling factor:

Montgomery's Modular Multiplication Algorithm:

```
R := 0 ;
For i := 0 to s-1 do
Begin
    R  := R + a_i×B ;
    q_i := (-r_0 m_0^-1) mod r ;
    R  := (R + q_i×M) div r ;
End
```

Output: $R \equiv (A \times B \times r^{-s}) \bmod M$ with $0 \leq R < B+M$

Here r^{-s} is interpreted as the inverse of $r^s \bmod M$. The digit products such as $a_i \times B$ are generated over s cycles by using the multiplier to compute each digit by digit product $a_i \times b_j$ for $0 \leq j < s$ from least to most significant digit of B, propagating carries on the way so that a non-redundant representation can be used.

The digit multiplier is usually an 8-, 16- or 32-bit multiplier, and the modulus size n is almost invariably a multiple of this. It follows that the output R, bounded by $B+M$, can be expected to exceed s digits frequently. The overflow digit or bit can be removed by an extra conditional subtraction or, less efficiently, by computing with $s+1$ digits [18, 6]. Some implementations may desire an output less than M, and those also must include a conditional subtraction.

4 A Timing Attack

In the timing attack [20], this conditional subtraction was assumed to take place as necessary after each long integer multiplication, whether as a result of imposing an upper bound of s digits or of M. By observation, the conditional subtraction is typically required for between 10% and 25% of multiplications in the case of the digit bound, depending upon how close M is to its maximal size within its bit length bounds. This frequency is very helpful for an attacker because if he can correctly detect the conditional subtraction with any probability greater than $\frac{1}{2}$ then, with sufficient observations, he can distinguish operations for which the probabilities of a subtraction are different. Assuming identical distributions for the inputs, the probabilities for squares and random multiplications *are* different, and so the attacker can eventually tell them apart. This means that the standard square-and-multiply exponentiation algorithm is unsafe under the assumed security model. Each multiplication corresponds to a bit which is set in the exponent. Hence the ones in the secret exponent can be determined.

Let us elaborate. Various extra operations involving resetting counters and reading or writing data etc. are carried out between the long integer operations of an exponentiation, making the division points between them fairly clear in a power trace [13]. Consequently, any variation in the number of clock cycles is clearly distinguishable. Even with preventative measures in place, we might

reasonably assume the attacker is able to distinguish the conditional subtractions if he has sufficiently sensitive equipment.

We need also to assume that the same exponentiation scheme, say square-and-multiply, is used repeatedly with the same exponent. Then, after a number of observations of exponentiations, the attacker can determine approximate probabilities for the conditional subtraction after each long integer operation in the exponentiation.

Inspection of Montgomery's algorithm shows that the magnitude of the output is essentially determined by the magnitude of the inputs. For example, the most significant output digit is $a_s \times b_s$ plus a component which is essentially random because of the multiple of M which has been added into it. In fact, the output lies between ABr^{-s} and $ABr^{-s}+M$. Under the reasonable assumption that outputs are uniformly distributed over this interval as A and B vary, one can take $ABr^{-s}+M/2$ as the expected output. Then for a given distribution of A and B, it is possible to calculate the probability of this output exceeding s digits, i.e. the probability of the conditional subtraction occurring.

There are two cases to consider in the square-and-multiply algorithm: the random, independent inputs to a multiplication and the equal inputs to a square. In [20], integrating the product of the density functions for A and B over appropriate ranges provided an expression with a coefficient of $\frac{1}{4}$ for the probability of a conditional subtraction, whilst integrating the square of the density functions for A provided a similar expression but with a coefficient of $\frac{1}{3}$. In the case of M being close to its maximal value, this showed that conditional subtractions can be expected in about $\frac{1}{4}$ of all multiplications, but in around $\frac{1}{3}$ of all squares. Thus the attacker can be expected to distinguish them and obtain the secret exponent very quickly if his equipment is good enough.

In essence this argument is sound, but the detail is less easy to provide accurately. For example, each multiplication may have the original ciphertext C as one input and this may not be distributed in a known manner. Furthermore, the other input will be the output from a previous squaring and therefore not be distributed in the same way as the output from a random product because of the more frequent conditional subtractions just described. Indeed, if an exponent bit is zero, the next multiplication will have an input which is a fourth power. This will have yet another distribution. Thus, the previous history of the inputs to a multiplication or squaring affects the liklihood of the conditional subtraction taking place. However, the more recent the operation, the stronger its influence on the conditional subtraction. Hence, unless the attacker is extremely unlucky, he should manage to determine each exponent bit precisely because the frequencies of subtractions should be clearly different. Furthermore, he can run simulations on his estimate for the exponent and make adjustments if the frequencies are different from what he measures on the card. Only the most significant bits of M have a significant effect on the probabilities for a given history of the arguments. Under the assumption that the exponent is unchanged on successive observations, and the public encryption exponent E is a standard, small choice such as 3, D will be a known, simple, linear function of $\phi(M)$. As $\phi(M)$ and M share

the same top bits, the tops bits of M can be obtained from those of D and vice versa. Hence, after trying possibilities for the topmost bits of D exhaustively if necessary, the attacker will very soon be able to determine subsequent exponent bits reliably by comparison with a simulation if any unusual circumstances arise.

Such variations in distributions allow a similar attack on m-ary exponentiation, at least for the small m which are the only practical values for a smartcard. However, the most distinctive distributions are obtained by partitioning observations into subsets according to whether or not a conditional subtraction has occurred for a particular operation. In particular, for $m = 4$ the values of $C^{(1)}$, $C^{(2)}$ and $C^{(3)}$ are pre-computed using Montgomery multiplications, the first being the result of introducing the Montgomery scaling factor [18]. So eight different subsets can be formed. These behave sufficiently distinctly for a different but characteristic vector of frequencies to be obtained for different exponent digits. According to whether the exponent digit has value 0, 1, 2 or 3, no multiplication or a multiplication by $C^{(1)}$, $C^{(2)}$ or $C^{(3)}$ takes place. The vector of frequencies determines which digit is correct, as is illustrated by the figures in [20]. Hence the secret exponent D can be reconstructed by the attacker.

5 Increased Key Length

Suppose the key length is increased. Does the attack become more or less difficult? The value of s is increased, but similar computations with density functions are going to determine similar constants again, namely $\frac{1}{4}$ for multiplications and $\frac{1}{3}$ for squares. Hence, it should be equally possible to determine whether or not an operation is a square or a multiply if occurrences of conditional subtractions can be detected equally easily. Of course, $O(s^2)$ clock cycles have to be counted for each long integer multiplication, but the attacker can reasonably be expected to be able to observe individual clock cycles and hence spot the variations that arise from conditional subtractions, which take $O(s)$ cycles to perform. So we should assume conditional subtractions are observed with an accuracy independent of the key length, and that the same is true for determining squares and exponent digits.

However, there are more exponent digits to determine. Suppose each is chosen correctly with probability p. Then p^t is the probability of determining D correctly, and this clearly decreases as the exponent size t is increased. More precisely, the frequency of subtractions for a particular operation enables an accurate estimate for the correctness of the deduced exponent digit to be made. If errors are made, this enables alternative choices to be ranked and the most likely tested for correctness. Overall, given an equal number of observations, exponents of any length can be expected to yield similar distributions of frequency counts. Hence, if p is the (geometrical) average probability of having a correct exponent digit, p^t will indeed provide the probability of the whole exponent being correct. In particular, doubling the key length will square the probability of correctness, and square the average number of different exponents that have to be considered before the correct one is found. Consequently, the difficulty is still exponential in

the length of the key, although when p is close to 1 there is no longer any reason to expect that an infeasible amount of computing is still required to break the card.

Needless to say, this emphasises the need for countermeasures. There is a standard and easy solution which prevents this attack, to found in [9]. For each exponentiation a fresh 32-bit random multiple of $\phi(M)$ is added to D and the result used as the exponent instead of D. The attacker is then unable to combine successive observations in the way required required for the attack. For typical exponent lengths the extra cost is under 10% plus the time for generating the random word.

Incidentally, one might also ask what effect there is in increasing m if the m-ary exponentiation scheme is used. The number of observation subsets provided by the pre-computations is $2^(m-1)$. Thus the vector of frequencies provides much more information as m increases, even if the total number of observations is fixed. So individual digits values should be easier to deduce correctly if all other parameters, such as key length, are kept constant.

6 A DPA Attack

The other attack considered here is one based on differential power analysis (DPA) [19]. Power use varies with the amount of switching activity in a circuit. In the case of a multiplier, simulations by the author have confirmed that the average number of gates switched from a random initial state during a multiply-accumulate operation is linear in the sum of the Hamming weights of the inputs. With a combination of power traces and EMR measurements from a carefully positioned probe [5], we will assume that the attacker can obtain some data, however minimal, which is related to the sum of the Hamming weights of these inputs. His problem is to combine these in a manner which reveals the Hamming weights with sufficient accuracy for him to deduce the digits of the exponent.

In m-ary exponentiation, the pre-computed powers $C^{(i)}$ are used as multipliers every time the digit i appears in the exponent. As with the previous attack, identifying this multiplier leads to a determination of the secret exponent D. Consider traces from a single, long integer multiplication $A \times C^{(i)}$ during the observation of one exponentiation. The calculation requires every product of digits $a_j \times c_k$ to be computed. For each k, the power or EMR traces are averaged as j ranges over its s values. In general, the digits of A are sufficiently random for this averaging process to provide a trace which is reasonably indicative of the Hamming weight of c_k. As key length s increases, this average *improves*. Concatenating these averaged traces together provides a trace which is characteristic of the particular $C^{(i)}$ which has been used. Roughly speaking, it corresponds to a vector of the Hamming weights of the digits of $C^{(i)}$. The Euclidean distance between any pair of such vectors for $C^{(i)}$ and $C^{(i')}$ with $i \neq i'$ has a very small variance compared to its average, so that the pair are almost invariably clearly separated. Thus, the attacker can expect to distinguish the traces he has formed from different multipliers $C^{(i)}$ and also to identify those formed using the

same multiplier $C^{(i)}$. From different multiplications during the exponentiation, he groups together traces which are close together under the Euclidean metric. The members of each such group should correspond to the use of the same exponent digit, and the different groups should correspond to different exponent digits. Comparing every pair of traces provides confirmation of such decisions already made because the traces will usually be clearly separated or clearly close together. The squaring operations can be identified in a similar way: they turn out to have traces which are not close to any others. Moreover, a fixed pattern of multiplications must be followed: a repetition of $\log_2 m$ squarings followed by an optional multiplication by some $C^{(i)}$. There are $(m-1)!$ ways of associating the correct digits to each multiplication to obtain the exponent D, and each of these can be tried in turn for correctness, but it is also possible to determine the correct association from the pre-multiplications since $C^{(i)}$ is used as a multiplier to form $C^{(i+1)}$. *From a single exponentiation* the attacker can therefore hope to deduce the exponent D.

As in the timing attack, the distance between traces can be used to predict the accuracy of the assignment of an exponent digit. The most likely alternatives can then be constructed and tested until the correct value of D is ascertained.

7 Increased Key Length

The effect of increasing the key length t in this attack has already been hinted at. Increasing t, and hence s, improves the trace averaging process. Furthermore, the concatenated traces have a length proportional to s, and therefore provide more data for comparison with other traces. This makes it easier to identify traces corresponding to the same exponent digit and distinguish those derived from different digits. Further still, the larger number $O(t^2)$ of pairs of traces for comparison increases the probability that positions corresponding to equal exponent digits will be correctly associated.

Overall, the effect of doubling the key length, say, is to double the length of the trace as well as increase its accuracy. This alone can be expected to increase the probability p of a correct association for a single digit to at least $\sqrt{p}$, so that, following the same argument as for the timing attack, the total probability p^t of deducing the correct exponent is not increased. Taking into account the larger number of pairs of traces which corroborate earlier choices, it is clear that the longer key length may well *improve* the chances of breaking the card.

Figures from a simulation of 8-ary exponentiation with a 32-bit multiplier and various key lengths are reproduced from [19] and extended in Table 1. The simulation was based on the number of gates which were switched in the multiplier after random initialisation and this was used to form traces of gate switch counts. The averages and standard deviations for the distances between traces corresponding to the same multiplier and to different multipliers were computed for 5×10^5 pairs in each column. Comparing two traces corresponding to the same multiplier, the average separation is proportional to the key length, as one would expect, as is the standard deviation. However, the average separation be-

tween traces representing different multipliers increases much faster: threefold per doubling of key length over the given range. In contrast, the standard deviation again appears to be linear in the key length. Consequently, for 256-bit keys the two averages are about 4.4 SDs apart, this rises to 6.0 for 512-bit keys, and then to 11.4 for 1024-bit keys. A quick look at normal distribution tables shows that the probability of being closer to the correct average goes up much faster than the square root as key length is doubled: $p_{256} = 0.9861$, $p_{512} = 0.99865$ and $p_{1024} = 1 - 0.57{\times}10^{-8}$ satisfy $p_{256} < {p_{512}}^2$ and $p_{512} < {p_{1024}}^2$. This probability is then further improved because of the increased number of pairs of traces that can be compared. The other columns of the table exhibit the same behaviour. Thus, at least in the simulation, it becomes much easier to deduce *all* exponent digits as the key length increases. Indeed, the percentage of errors in the simulation quickly decreases to zero over the range of the table. This confirms that it should become easier to distinguish distinct multipliers as key length increases, with a sufficient improvement to enable the whole key to be recovered much more easily.

Length M (bits)	96	128	192	256	384	512	768	1024
Av btwn same	470	798	1263	1529	2366	3750	4501	6246
SD btwn same	257	347	681	885	1403	2386	2535	3612
Av btwn diff	1224	2120	4973	5890	11753	17896	32594	53070
SD btwn diff	442	585	1276	1108	2412	2279	4646	4581
%age errors	57.24	40.20	5.307	0.9284	0.1155	0.2819	0.0000	0.0000
SDs btwn avs	2.16	2.84	3.79	4.4	4.9	6.0	7.8	11.4
p_t	0.8596	0.9220	0.9710	0.9861	0.9929	0.9986	.99995	0.999..

Table 1. Gate Switch Statistics for 32-bit multiplier with $m = 8$.

8 Counter-measures

The counter-measure proposed for the timing attack, namely randomly varying the exponent on each exponentiation, does not work here because this DPA attack is applied to a single exponentiation. However, it relies on a known order for the digit products being formed as well as the repeated use of the same multipliers, which indicate the exponent digits. Hence the digit products could be randomly re-ordered, although the cost of this is substantial because of losing the convenience of sequential carry propagation. Also, the square-and-multiply algorithm could be employed, but in the form which processes the digits starting from the least significant. This avoids re-using the same multiplier, but requires more writing of results to memory. Such writes may reveal whether or not a multiplication is being performed, and hence provide the bits of the exponent.

However, the best counter-measure may just be to pay for a larger multiplier. This reduces the number of digits over which averages are taken and reduces the number of concatenated traces, i.e. it reverses the effect of increased key

length on the traces, but allows t to be increased. Hence the probability of a successful attack can be reduced. Moreover, with larger numbers of words sharing the same Hamming weight, it is no easier to use the Euclidean metric to separate the different multipliers. Further, one might use two multipliers in parallel. Montgomery's modular multiplication algorithm naturally uses two. The power used by one might successfully shield observation of the power used by the other, and an EMR probe may not be precise enough to distinguish between them. Thus safety can be bought, but at a price.

9 Conclusion

Two attacks on smartcards have been outlined, one a timing attack and the other a power analysis attack. The effect of increasing the key length was studied for each, and it provides improved security against the timing attack. However, every indication shows that the DPA attack becomes easier since sufficiently more data is available to guide each decision. Current standard algorithmic countermeasures do not provide a very satisfactory solution to this problem. However, investing in one or more larger multipliers on the smartcard certainly makes the attack more difficult.

References

1. R. M. Best, *Crypto Microprocessor that Executes Enciphered Programs*, U.S. Patent 4,465,901, 14 Aug. 1984.
2. J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater & J.-L. Willems, *A practical implementation of the Timing Attack*, Proc. CARDIS 1998, Lecture Notes in Computer Science, **1820**, Springer-Verlag, 2000, pp. 175–190.
3. W. Diffie & M. E. Hellman, *New Directions in Cryptography*, IEEE Trans. Info. Theory, **IT-22**, no. 6, 1976, pp. 644–654.
4. T. El-Gamal, *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. Info. Theory, **IT-31**, no. 4, 1985, pp. 469–472.
5. K. Gandolfi, C. Mourtel & F. Olivier, *Electromagnetic Analysis: Concrete Results*, Cryptographic Hardware and Embedded Systems (Proc CHES 2001), Ç. Koç, D. Naccache & C. Paar editors, Lecture Notes in Computer Science (*to appear*), Springer-Verlag, 2001.
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, pp. 293–301.
7. D. E. Knuth, The Art of Computer Programming, vol. 2, *Seminumerical Algorithms*, 2nd Edition, Addison-Wesley, 1981, pp. 441–466.
8. Ç. K. Koç, *Analysis of Sliding Window Techniques for Exponentiation*, Computers and Mathematics with Applications, **30**, no. 10, 1995, pp.17–24.
9. P. Kocher, *Timing attack on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Proc. Crypto 96, N. Koblitz (editor), Lecture Notes in Computer Science, **1109**, Springer-Verlag, 1996, pp. 104–113.

10. 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, pp. 388–397.
11. M. G. Kuhn *Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP*, IEEE Transactions on Computers, **47**, No. 10, October 1998, pp. 1153–1157
12. R. Mayer-Sommer, *Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards*, Cryptographic Hardware and Embedded Systems (Proc CHES 2000), C. Paar & Ç. Koç editors, Lecture Notes in Computer Science **1965**, Springer-Verlag, 2000, pp. 78–92.
13. T. S. Messerges, E. A. Dabbish, R. H. Sloan, *Power Analysis Attacks of Modular Exponentiation in Smartcards*, Cryptographic Hardware and Embedded Systems (Proc CHES 99), C. Paar & Ç. Koç editors, Lecture Notes in Computer Science **1717**, Springer-Verlag, 1999, pp. 144–157.
14. V. Miller, *Use of Elliptic Curves in Cryptography*, Proc. CRYPTO '85, H. C. Williams (editor), Lecture Notes in Computer Science **218**, Springer-Verlag, 1986, pp. 417–426.
15. P. L. Montgomery, *Modular Multiplication without Trial Division*, Mathematics of Computation, **44**, no. 170, 1985, pp. 519–521.
16. R. L. Rivest, A. Shamir & L. Adleman, *A Method for obtaining Digital Signatures and Public-Key Cryptosystems*, Comm. ACM, **21**, 1978, pp. 120–126.
17. W. Schindler, *A Timing Attack against RSA with Chinese Remainder Theorem*, Cryptographic Hardware and Embedded Systems (Proc CHES 2000), C. Paar & Ç. Koç editors, Lecture Notes in Computer Science **1965**, Springer-Verlag, 2000, pp. 109–124.
18. C. D. Walter, *Montgomery Exponentiation Needs No Final Subtractions*, Electronics Letters, **35**, no. 21, October 1999, pp. 1831–1832.
19. C. D. Walter, *Sliding Windows succumbs to Big Mac Attack*, Cryptographic Hardware and Embedded Systems (Proc CHES 2001), Ç. Koç, David Naccache & Christof Paar (editors), Lecture Notes in Computer Science, Springer-Verlag, *to appear*.
20. 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, pp. 192–207.
21. *Digital Signature Standard* (DSS), FIPS 186, http://csrc.nist.gov/publications/, US National Institute of Standards and Technology, May 1994.
22. *Data Encryption Standard* (DES), FIPS 46-3, http://csrc.nist.gov/publications/, US National Institute of Standards and Technology, October 1999.
23. *Advanced Encryption Standard* (AES), FIPS draft, http://csrc.nist.gov/ publications/, US National Institute of Standards and Technology, 2001.
24. *Index of National Security Telecommunications Information Systems Security Issuances*, NSTISSC Secretariat, US National Security Agency, 9 January 1998.