Distinguishing Exponent Digits by Observing Modular Subtractions

Colin D. Walter* & Susan Thompson

Datacard platform[7] seven
6th-8th Floors, 1-2 Finsbury Square, London EC2A 1AA, UK
www.platform7.com

Abstract. We analyse timing variations in an implementation of modular multiplication which has certain standard characteristics. This shows that squarings and multiplications behave differently when averaged over a number of random observations. Since power analysis can reveal such data, secret RSA exponents can be deduced if a standard square and multiply exponentiation algorithm is used. No knowledge of the modulus or input is required to do this. The technique generalises to the m-ary and sliding windows exponentiation methods since different multipliers can be distinguished. Moreover, only a small number of observations (independent of the key size and well under 1k) are required to perform the cryptanalysis successfully. Thus, if the modular multiplication algorithm cannot be made any safer, the exponent must be modified on every use.

Key words: Exponentiation, modular multiplication, Montgomery multiplication, RSA cryptosystem, m-ary method, sliding windows, timing attack, power analysis.

1 Introduction

Smart cards may contain sensitive data, such as private RSA keys [7], which may be of great value to an attacker if they can be retrieved. These may well be used for all authentication and key exchange processes, and so must not be compromised. However, we illustrate how one likely source of timing variation during modular multiplication can be exploited to reveal such keys with very few observations.

Kocher [5] wrote one of the earliest, relevant, publicly available documents on time-based attacks and he relies for success on knowing the plaintext inputs. The causes of time variations are explicit conditional statements in the software, and implicit conditionals introduced by the compiler or hardware, most usually in the cause of optimisation. Skipping a multiplication by 0 is a typical example of the

* contact address: Computation Department, UMIST, Manchester, M60 1QD, UK, www.co.umist.ac.uk

latter which causes unexpected time variation. An example of the former is that the standard modular multiplication algorithms make conditional subtractions of the modulus to keep the result within a fixed upper bound. It is this extra subtraction that is the subject of study here. Dhem *et al.* [2] provided practical details for using it in Kocher's attack to obtain RSA keys. They repeatedly assume the next unknown exponent bit is 1 and partition the known plaintext inputs into two sets according to whether or not the extra subtraction occurs for them in the corresponding multiplication of the exponentiation routine. With enough observations, if different average times occur for the two sets, the bit must be 1 and otherwise it is 0. For 512-keys 300,000 timings must be collected for the attack to succeed.

Recent independent work at Platform[7] Seven [1] and by Schindler [8] has provided theoretical justification for this. Both show that in Montgomery's modular multiplication algorithm [6], the need for a final subtraction to obtain a result less than the modulus is different for squares and multiplications. Borovik and Walter [1] used this in the way described here to read secret RSA exponent bits directly using unknown plaintexts. Schindler [8] used it to attack implementations which make use of the Chinese Remainder Theorem to reduce the arithmetic. However, Schindler's is a chosen plaintext attack.

Here we develop the attacks to a much wider setting and, in particular, to unknown or blinded inputs with unknown modulus and more general exponentiation algorithms. The paper commences with theoretical explanation of the observed frequency of modular subtractions, enabling predictions about the average behaviour of squares and multiplies. This provides a much clearer picture of how to use timing measurements to reveal a secret RSA exponent. A little more strongly than Schindler, it is assumed that power, timing, or other measurements during each exponentiation are clear enough to enable the presence or absence of an extra modular subtraction to be detected for *each* individual multiplication. For each multiplication or squaring in an exponentiation scheme, the frequency of subtractions can then be computed for a set of observations and used to differentiate between the two operations.

If the usual square and multiply exponentiation algorithm has been used, this process yields the exponent bits immediately. Indeed, straightforward statistics can be applied to noisy data to deduce how many observations need to be made to obtain the exponent with a given probability. For clean data, this number turns out to be so small as to make the smart card totally insecure, and therefore useless, unless adequate counter-measures are employed.

By carefully selecting appropriate subsets of observations, the same techniques can be applied to any sequence of multiplications which only uses multipliers from a small set, in order to identify which multiplier has been used. As a result, the usual m-ary [3] or sliding window methods [4] of exponentiation are also vulnerable to this attack. For $m = 4$, under 1000 observations suffice. Moreover this result is *independent* of the key length because the exponent digits are determined independently, not sequentially.

The conclusion is clear: if sufficient timing information can be gleaned, then either such numerous conditional modular adjustments must be avoided (as they can be – e.g. *see* [9]) or the exponent must be adjusted before each new decryption in order to confound the averaging process [5].

2 Timing Variations in Modular Multiplication

2.1 Initial Assumptions

For the purpose of this paper we consider a generic multi-precision implementation of Montgomery multiplication [6] used in the context of an RSA decryption, but similar timing attacks can be mounted against other modular multiplication algorithms which display the same weakness as is exploited here.

We assume that arithmetic is based on an m-bit architecture. Hence all numbers are presented in radix $r = 2^m$. Let k be the fixed number of digits used to represent the arguments and intermediate results of the exponentiation. Then $l = mk$ is the number of bits in such numbers. For convenience and because it is to be expected, we will assume the modular multiplication algorithm performs l addition cycles so that the Montgomery scaling constant is $R = 2^l$ [10]. It is natural to use as large a modulus N as possible, and so

– *We assume that $R/2 < N < R$.*

This is perhaps the major drawback of many implementations, because it forces a conditional modular subtraction to be made if an overflow bit is to be avoided.

2.2 Analysis of the Modular Reduction

Let R^{-1} be the integer uniquely determined by the conditions $R{\cdot}R^{-1} \equiv 1 \bmod N$ and $0 < R^{-1} < N$. This exists because R is a power of 2, ensuring that it has no non-trivial common factor with the odd modulus N. Given non-negative integers $A < R$ and $B < R$, the main loop of Montgomery multiplication returns a number

$$M \;\equiv\; ABR^{-1} \bmod N$$

in the range $0 \le M < B{+}N$. Hence an extra subtraction of N or even of $2N$ may be required to get a residue less than N because $B < 2N$. In particular, this subtraction might be deemed worthwhile to avoid the result overflowing into an extra digit position.

In this paper we perform a cryptanalysis based on the conditions under which such extra subtractions are performed at the end of each modular multiplication. Since $M{-}N < B < R$, we concentrate on the version of the algorithm for which the reduction is made at most once to a level below R:

– *We assume the modular multiplication algorithm includes a final conditional statement for modular subtraction, namely*

$$M \;:=\; \begin{cases} M & \text{if } M < R \\ M{-}N & \text{if } M \ge R\,. \end{cases}$$

This reduction is easier and faster to implement in hardware than obtaining the least non-negative residue, and it suffices for RSA exponentiation which employs repeated modular multiplication. However, the version of the algorithm with the modular reduction to a level below N (as in [8]) can be analysed analogously and, from the practical point of view, the analysis yields similar results in exactly the same way. Of course, obtaining such a tight bound, i.e. the minimal non-negative residue mod N, is computationally more expensive and so is often performed only at the very end of the exponentiation.

Hardware limitations require that both multiplicands A and B and the modulus N are smaller than R. So, written to the base r, they have the forms

$$\begin{aligned} A &= (a_{k-1}a_{k-2}\ldots a_1a_0)_r, \\ B &= (b_{k-1}b_{k-2}\ldots b_1b_0)_r \qquad \text{and} \\ N &= (n_{k-1}n_{k-2}\ldots n_1n_0)_r \end{aligned}$$

where $0 \le a_i < r$, $0 \le b_i < r$ and $0 \le n_i < r$.

Let $n' := (r-n_0)^{-1}$ mod r. Then the Montgomery multiplication routine runs as follows:

$$\begin{array}{l} S_0 := 0 \ ; \\ \textbf{for } i := 0 \textbf{ to } k-1 \textbf{ do} \\ \qquad S_{i+1} := \{S_i + a_iB + ((S_i + a_iB)\cdot n' \bmod r)\cdot N\}/r \\ \textbf{end} \end{array}$$

Here $(S_i{+}a_iB)$ mod r is given by the rightmost digit of $S_i{+}a_iB$ to the base r which is, of course, equal to $(s_{i0}{+}a_ib_0)$ mod r. S_{i+1} is clearly always an integer.

Notice that, by induction,

$$r^iS_i \;\equiv\; (a_{i-1}\ldots a_0)_r\cdot B \bmod N$$

and we can also prove by induction on i that $S_i < B + N$. Indeed $S_0 = 0$ gives us the basis of induction, and

$$\begin{aligned} 0 \;\le\; S_{i+1} \;&=\; \frac{1}{r}S_i + \frac{a_i}{r}B + \frac{(S_i{+}a_iB)_0\cdot n' \bmod r}{r}\cdot N \\ &<\; \frac{1}{r}(B{+}N) + \frac{r-1}{r}B + \frac{r-1}{r}N \\ &=\; B{+}N \end{aligned}$$

Hence

$$S_{k-1} \;\equiv\; ABR^{-1} \bmod N$$

and

$$S_{k-1} \;<\; B{+}N \;<\; R{+}N \;<\; 2R$$

Note the asymmetry between multiplicand A and multiplier B in this bound. To return a value of $M = ABR^{-1}{+}\kappa N$ which is strictly less than R, we need to set

$$M \;:=\; \begin{cases} S_{k-1} & \text{if } \ S_{k-1} < R \\ S_{k-1} - N & \text{if } \ S_{k-1} \ge R \end{cases}$$

This last adjustment is a possible cause of time variations in modular multiplications. It might be avoided by performing the subtraction whether it is necessary or not, and then selecting one of the two results. However, timing variations may still creep in here because of compiler optimisations or because a different number of register movements is performed. Be warned!

– *We assume that timing or other variations enable all or almost all occurrences of this final subtraction to be observed.*

Notice now that the value of S_{k-1} has very strong dependence on B through the middle term of the expression for it:

$$S_{k-1} \quad = \quad \frac{1}{r}S_{k-2} + \frac{a_{k-1}}{r}B + \frac{(S_{k-2}+a_{k-1}B)_0 \cdot n' \bmod r}{r}N$$

So one has to expect much more frequent “long” multiplications (that is, multiplications which involve the final modular adjustment) for larger values of B. These can be expected particularly as N approaches R.

2.3 Analytical Approximation

The modular adjustment happens when the random variable $\sigma = S_{k-1}/R$ is greater than or equal to 1. Then σ can be expressed in terms of other random variables, namely

$$\sigma \;=\; \alpha{\cdot}\beta + \nu + \gamma$$

where

$$\begin{aligned}
\alpha &= \frac{a_{k-1}+\frac{1}{2}}{r} \approx \frac{A}{R}\,, \\
\beta &= \frac{B}{R}\,, \\
\nu &= \frac{(S_{k-2}+a_{k-1}B)_0 \cdot n' \bmod r}{r} \cdot \frac{N}{R} + \frac{N}{2rR}\,, \\
\gamma &= \frac{S_{k-2}}{rR} - \frac{B+N}{2rR}
\end{aligned}$$

are random variables distributed in some way over the intervals (0,1), [0,1), $(0,N/R)$ and $(-\frac{1}{r},\frac{1}{r})$ respectively. Let us investigate the distributions that these random variables might have in the context of exponentiation. For this,

– *We assume that A and B are uniformly distributed* mod N.

This may not hold for the initial one or two operations of an exponentiation because of the manner in which the initial input is formed. But, the whole value of modular exponentiation as an encryption process is its ability to give what appears to be a uniform, random output mod N no matter what the input has been. Since 3 is accepted as a suitable encryption exponent, we can reasonably assume that after two or three multiplicative operations, the inputs to further operations in an exponentiation are close enough to being uniform modulo N for our purposes.

Since the coefficient

$$0 \leq \frac{(S_{k-2} + a_{k-1}B)_0 \cdot n' \bmod r}{r} \leq 1 - \frac{1}{r}$$

of N/R in the expression for ν is sufficiently randomised by modulo r arithmetic, we can assume that ν is independent of α and β and is uniformly distributed in the interval $(0, N/R)$. (It is easy to deduce from earlier multiplications that this is the case even if B has been shifted up to make the computation easier.) We will argue that A and B are piecewise uniformly distributed on their intervals so that the same is true for α and β. Clearly α and β are not independent for squaring operations since then $A = B$, but we will justify that they are essentially independent for almost all of the multiplications.

We will now prove that γ is smaller than $\alpha\beta+\nu$ by a factor of order $\frac{1}{r}$ so that its contribution to σ may be neglected. Since A is assumed uniformly distributed mod N, for non-small i, S_i can also be expected to be randomly and uniformly distributed mod N because its residue class is determined by a (large) suffix of A times B. As S_i belongs to the interval $[0, B{+}N)$ but can achieve both end points, and the added multiple of N is essentially random, the most reasonable expectation is that S_i is piecewise uniformly distributed over the three subintervals $[0, B)$, $[B, N)$ and $[N, B{+}N)$ with probabilities $\frac{1}{2N}$, $\frac{1}{N}$ and $\frac{1}{2N}$ respectively. This leads to an average of $\frac{1}{2}(B{+}N)$ for S_i and therefore to an expected average of 0 for γ.

Consider the case when $B{+}N < R$. Then $S_{k-1} < B{+}N$ ensures that no final subtraction takes place. Hence, under the uniformity assumption mod N, the distribution of the output will be identical to that of S_{k-1} given above. So, such output provides a mean of $\frac{1}{2}(B{+}N)$, which is less than $\frac{1}{2}R$. Otherwise, to preserve uniformity mod N, when the subtraction takes place the output distribution will be close to uniform on each of the subranges $[0, R{-}N)$, $[R{-}N, N)$ and $[N, R)$, yielding instead an average of $\frac{1}{2}R$ for the output. Thus,

- *For a given input B, the output from a modular multiplication is approximately piecewise uniform on the interval $[0, R)$. For $B{+}N < R$ the intervals of uniformity depend on B. In both cases there are three intervals with non-zero probabilities $\frac{1}{2N}$, $\frac{1}{N}$ and $\frac{1}{2N}$ respectively.*

By the above, if modular multiplier outputs are used for the inputs A and B of a subsequent multiplication, then their average values match or exceed $\frac{1}{2}(B{+}N)$, which is bounded below by $\frac{1}{4}R$. Thus we obtain lower bounds of at least $\frac{1}{4}$ for each of α and β. So, α and β are at least $\frac{r}{4}$ times larger than γ on average. Hence, we can ignore the contribution of γ providing:

- *We assume that the radix r is not too small.*

Commonly used bases such as $r = 2^8$, 2^{16} and 2^{32} are quite acceptable here. From the above, we can expect that the statistics for final adjustments in the

Montgomery multiplication

$$(A, B) \longrightarrow A \otimes_N B \equiv ABR^{-1} \bmod N$$

are sufficiently close to the statistics of occurrences of the subtraction in the product

$$\alpha \otimes \beta = \begin{cases} \alpha\beta + \nu & \text{if } \alpha\beta + \nu < 1 \\ \alpha\beta + \nu - \rho & \text{if } \alpha\beta + \nu \geq 1 \end{cases}$$

where $\rho = N/R$. The radix r is large enough for the discreteness of the original problem to make only a marginal difference to the calculations if we substitute *continuous* random variables for the discrete ones: the relative errors will invariably be bounded above by at most about $\frac{1}{r}$ which, by assumption, is small.

2.4 Heuristic Estimates for Multiplications

In order to get some intuition regarding the behaviour of Montgomery multiplication, let us assume, like Schindler [8], that

– *α is uniformly distributed on* $(0, 1)$

The previous section clearly shows that this is a simplification. The average output of the modular multiplier is less than $R/2$ so that the distribution of α over $[0, R)$ cannot be uniform. However, providing N is close to R, such an assumption is only slightly frayed at the edges.

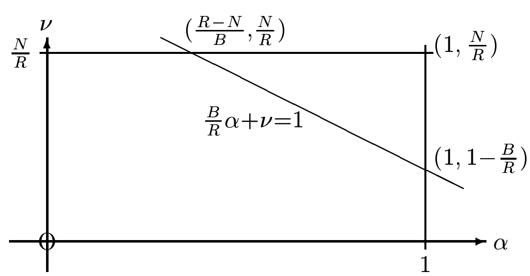

Fig. 1. Computation of $P(\alpha\beta+\nu \geq 1)$.

Suppose β has the fixed value $\beta = B/R$. (So this is *not* a squaring.) The modular adjustment takes place when the point (α, ν) belongs to the upper right corner cut from the rectangle $[0, 1]\times[0, N/R]$ by the line $\alpha\beta+\nu = 1$ (*see* Figure 1). The probability of this event is the ratio of the area of the triangle to that of the rectangle, namely

$$P_{\text{mult}}(B) = P(\alpha\beta+\nu \geq 1) \approx \begin{cases} 0 & \text{if } N+B < R \\ \frac{(B+N-R)^2}{2BN} & \text{if } N+B \geq R. \end{cases}$$

As expected, the reductions occur more frequently the larger $B+N$ is, and, in particular, they normally occur in a sizeable proportion of all observations.

It is possible to obtain a more precise formula for P_{mult} as a function of B using the piecewise uniform probability function described in the previous section. However, this detail is unnecessary for the attack which we describe. It is sufficient to note that when we select a set of observations involving smaller than average values of B then we can expect fewer subtractions to occur. This will happen, paradoxically, if such B are the outputs of previous modular multiplications for which the extra subtraction did *not* occur (since we saw the average was $\frac{1}{2}R$ after a subtraction, but only $\frac{1}{2}(B+N) < \frac{1}{2}R$ otherwise).

2.5 Probability Estimates for Multiplications & Squarings

With the same definitions as before, the probability of modular adjustment in a Montgomery multiplication of independent arguments is

$$P_{\mathrm{mult}} \;\approx\; P(\alpha\beta+\nu \geq 1) \;=\; \int_{1-N/R}^{1}\int_{(1-N/R)/x}^{1}\int_{1-xy}^{N/R} p(x,y,z)dzdydx$$

where p is the probability density function for $\alpha\times\beta\times\nu$. The randomising effect of raising to an odd power of at least 3 means that most operands in the multiplications of an encryption or decryption will be effectively independently distributed mod N. Hence, assuming this is the case, we could write p as a product of three functions of a single variable, representing the individual density functions for α, β and ν respectively. As noted above, ν is uniform on $[0,N/R]$. If we simplify by assuming $p_\alpha(x) = p_\beta(x) = 1$ then

$$\begin{aligned}
P_{\mathrm{mult}} &\approx \frac{R}{N}\int_{1-N/R}^{1}\int_{(1-N/R)/x}^{1}\int_{1-xy}^{N/R} dzdydx \\
&= \frac{R}{N}\int_{1-N/R}^{1}\left\{\frac{1}{2}x - (1-\frac{N}{R}) + \frac{1}{2}(1-\frac{N}{R})^2\frac{1}{x}\right\}dx \\
&= \frac{R}{4N}(1-(1-\frac{N}{R})^2) - (1-\frac{N}{R}) - \frac{R}{2N}(1-\frac{N}{R})^2\log(1-\frac{\mathrm{N}}{\mathrm{R}})
\end{aligned}$$

In the same way, the probability of a modular adjustment in a Montgomery square is

$$P_{\mathrm{square}} \approx P(\alpha^2+\nu \geq 1) \;=\; \int_{\sqrt{1-N/R}}^{1}\int_{1-x^2}^{N/R} p(x,y)dydx$$

where p is now the probability density function for $\alpha\times\nu$. Since α and ν are independent and ν is uniform on $[0,N/R]$, we can re-write this as

$$P_{\mathrm{square}} \;\approx\; P(\alpha^2+\nu \geq 1) \;=\; \int_{\sqrt{1-N/R}}^{1}\int_{1-x^2}^{N/R} p_\alpha(x)R/N dydx$$

$$= \frac{R}{N}\int_{\sqrt{1-N/R}}^{1}(\frac{N}{R}-1+x^2)p_\alpha(x)dx$$

Once more, we will simplify by assuming that A is uniformly distributed on $[0,R)$. Then $p_\alpha(x) = 1$, so that

$$P_{\text{square}} \;\approx\; 1-\frac{2R}{3N}\left(1-(1-\frac{N}{R})^{3/2}\right)$$

At the upper end of the range for N, namely N close to R, we see that the expression for the square is approximately $\frac{1}{3}$, while that for the multiplication is only about $\frac{1}{4}$. Hence squares can easily be distinguished from multiplications with independent arguments by the frequencies of the extra subtractions. Although the density functions become progressively more badly approximated by 1 as N decreases, the piecewise linear nature of the true density function can be used to obtain accurate formulae which demonstrate a similar difference for all values of N in the interval $(\frac{1}{2}R,R)$.

These formulae display a potentially useful dependence on N which might be exploited to deduce an approximate value for N from observing the actual value of P_{square} or P_{mult}. Moreover, if input A can be restricted in some way, the density function may be modified enough to provide detectable changes in P_{square} or P_{mult}.

For example, suppose the multiplicative operation Op_1 (square or multiply) generates the input A to the multiplicative operation Op_2 as part of some process, such as an exponentiation. Partition a set of observations of the process into two subsets, one for which Op_1 applies the extra adjustment and the other for which it does not. The study in a previous section shows the related density functions for A are sufficiently different to yield distinct averages for A and so will usually yield two different values for the frequencies of extra subtractions at the end of Op_2. This enables us to determine which multiplicative operation Op_1 has generated the argument used in Op_2. If the wrong operation Op_1 is selected, we expect a much lower difference between the density functions so that there is little difference between the observed frequencies for the two subsets.

3 Attacks on Exponentiation with Unknown Modulus

3.1 Unknown Plaintext Attack on the Square & Multiply Method

The standard square and multiply method of exponentiation uses the binary representation

$$e \;=\; \sum_{j=0}^{n} e_j 2^j$$

of the exponent e. It scans the bits of e in descending order and applies a Horner-style evaluation

$$X^e \;=\; ((\ldots((X^{e_n})^2X^{e_{n-1}})^2\ldots)^2X^{e_1})^2X^{e_0}.$$

Multiplication by X^{e_i} is performed conditionally whenever the bit e_i is 1.

When computing X^e mod N using modular Montgomery exponentiation we first replace X by XR mod N using a Montgomery multiplication by R^2. After that, the identity

$$AR \cdot BR \cdot R^{-1} \;\equiv\; ABR \bmod N$$

allows us to carry out the multiplication of (Montgomery) powers of XR until we get X^eR mod N. Montgomery multiplying this result by 1, we obtain the desired power X^e of X modulo N.

Thus, in this section and the next section all multiplications are understood as Montgomery multiplications modulo N. To make notation more transparent, we write $X \otimes_N Y$ instead of XYR^{-1} mod N and assume that the final modular reduction is done, when required, within the computation of $X \otimes_N Y$. Thus all the intermediate products are smaller than R and satisfy the conditions for the arguments of the modular Montgomery multiplication as set out in the previous section.

An immediate corollary from the statistical analysis of the previous section is that the probabilities of modular adjustments in a Montgomery square and a Montgomery multiplication are sufficiently large to make the adjustment detectable from only a few power traces, assuming that the timing differences can be seen. They should be noticeably more frequent in the positions of squares than those of multiplication. This makes it possible to read directly the bits of the exponent from the observational data since, except perhaps for the first multiplication, we can expect the arguments of each multiplication to be sufficiently independent of each other. So a timing attack is easy to perform on the square and multiply algorithm using unknown inputs.

3.2 Unknown Plaintext Attack on the m-ary Method

The m-ary method for the exponentiation $X \longrightarrow X^e$ [3, pp. 441–466] is a generalisation of the square and multiply method. The exponent e is expressed in terms of a base m,

$$e \;=\; \sum_{j=0}^{n} e_j m^j.$$

The powers X^i for $i = 1, 2, \ldots, m-1$ are precomputed and stored for multiplying into the partial result when required. The corresponding evaluation rule is

$$X^e \;=\; ((\cdots((X^{e_n})^m X^{e_{n-1}})^m \cdots)^m X^{e_1})^m X^{e_0}.$$

In the process, whenever the non-zero digit $e_j = i$ is encountered, the stored power X^i is multiplied in. For example, for $m = 4$, X, X^2 and X^3 are precomputed and stored.

The base m is usually a power of 2, so that computation of the m-th power of the partial result consists of several consecutive squarings. The sliding windows method [4] employs some recoding of the exponent and, among other things, performs a squaring when the next exponent bit is 0. This means that the even powers of X need not be stored. Now we describe our attack.

– *We assume that we do not know the modulus N of the exponentiation, nor have control or knowledge of plaintext inputs.*

Suppose we observe k runs of the exponentiation procedure involving different unknown plaintexts $A = A_1, A_2, \ldots, A_k$. These plaintexts should have been randomly generated in some manner but need not be uniformly distributed mod N. The initialisation process generates $X_i = A_i R \bmod N$ for input into the exponentiation. After this multiplication (by R^2), the numbers X_i will be more or less random mod N and belong to the interval $[0, R)$. Hence, after any necessary modular subtraction, the X_i will be distributed fairly uniformly over each of three sub-intervals according to the value of $R^2 \bmod N$.

As before, assume that we can detect whether or not the modular adjustment has taken place during the j-th multiplicative operation. If k is not too small, then these observations of essentially independent random encryptions enable fairly accurate determinations of the probabilities for the jth operation to require the modular adjustment. The previous section describes how these data can then be used to distinguish squares from multiplies.

– *Now assume also that the initially generated powers of X are used as the B inputs to the Montgomery modular multiplication process.*

Recall that the frequency of extra subtractions depends on the value of the B input. Because multiplications corresponding to the same exponent digit will make use of the same multiplier B, the expected frequency of extra subtractions will be the same for both multiplications whatever the set of observations. However, the randomness of these multipliers means that for different exponent digits, the multipliers will generally have different values and individually lead to different probabilities for an extra subtraction. So, if a subset of observations can be identified in which the multipliers corresponding to two exponent digit values have different properties, then potentially this will be reflected in different average frequencies for the extra subtractions in the multiplications corresponding to occurrences of these two digits.

In fact, it is possible to determine such observation subsets from any multiplication and, in particular, from behaviour during the initialisation stage when the powers X^i $(i = 1, 2, \ldots, m)$ are formed. For example, for an exponent digit i, partition the observations into two subsets according to whether or not the generating modular multiplication for X^i included an extra subtraction. We noted before that the average values for X^i must be different for the two sets. This will result in different frequencies for the extra subtraction when X^i is used as a multiplier in the exponentiation and when it is not. Hence, occurrences of exponent digit i should stand out. We illustrate this in Section 4.

Suppose this process has already been applied for each i to identify which exponent digits are most likely to be equal to i. Any pair of multiplications during the exponentiation can then be compared in the same way, providing a cross-check on the initial assignment.

Let M be the total number of multiplicative operations in the pre-computation and exponentiation combined. Then we could form a $k \times M$ observation matrix

$Z = (z_{ij})$ by writing $z_{ij} = 1$ if there was a modular adjustment in the j-th operation (a multiplication or squaring) of the i-th exponentiation, and $z_{ij} = 0$ otherwise. We have argued that there are strong dependencies between the columns Z_s and Z_t of the matrix Z if the s-th and t-th multiplication in the exponentiation routine are multiplications by the same power $X_i^{e_j}$ of X_i and which correspond to the same digit e_j in the exponent. Moreover, there are also strong dependencies between the column corresponding to the precomputations of $X_i^{e_j}$ and $X_i^{e_j+1}$ and the columns Z_s, Z_t corresponding to digit e_j. This, again, allows us to perform more advanced statistical analysis and deduce effectively the digits of the exponent from observation of adjustments.

3.3 The Danger of Signing a Single Unblinded Message

In this section we describe how it is possible to attack the exponent of a Montgomery based exponentiation, without the statistical analysis described in sections 2.3 to 2.5, if the modulus and a single plaintext input are known. This would be the case if an RSA signature were computed directly without use of the Chinese Remainder Theorem or appropriate counter-measures. The attack may be applied to both the square and multiply and m-ary methods although, for simplicity, only the square and multiply attack is described here.

Consider the single sequence of modular operations, squares and multiplies, formed by exponentiation of a *known* input A with secret exponent e and known modulus N. For any t, denote the most significant t bits of e by $e(t) = e_n e_{n-1}, \ldots, e_{n-t+1}$. Let $f(t)$ denote the number of modular operations (including precomputations) that result from using $e(t)$ as the exponent and let $Z = (z_j)$ be the observation vector indicating the extra subtractions. (We don't make use of any initial elements representing the precomputations.)

A binary chop on e may now proceed as follows. Suppose the t most significant bits of the exponent are known. We want to establish the value of the next bit. So far, $X = AR \bmod N$ and $Y = A^{e(t)}R \bmod N$ can be computed independently of the target device. Therefore Y is known and the observation vector $Z' = (z'_j)$ obtained from this exponentiation with $e(t)$ should match the first $f(t)$ elements of Z exactly.

To determine $e(t+1)$ compute the two Montgomery operations $Y := Y \otimes_N Y$ followed by $Y = Y \otimes_N X$. Extend the observation vector $Z' = (z'_j)$ by adding the two extra elements associated with these operations. Elements $f(t)+1$ should match in Z and Z' since the same square is being performed. If they don't match, a previous bit of e has been incorrectly assigned and backtracking is necessary to correct it [5]. Assuming the elements match and earlier bits were correctly assigned, if elements $f(t)+2$ do not match in both vectors then certainly $e_{n-t} = 0$ since different operations must be being performed for the two observation vectors. Otherwise, we assume $e_{n-t} = 1$ and continue.

Backtracking to fix incorrect bits is not expensive, and one simply has to choose exponent bits which are consistent with the vector Z. The average number of incorrect bits chosen before an inconsistency is discovered is very small. For simplicity, suppose that a subtraction occurs 1 in 4 times for both multiplications and squares, and that the numbers of subtractions required in two successive operations are independent. (As noted in section 2, this is close to what happens in reality.) Then the probability of a match between two elements is about $\frac{5}{8}$ when a previous bit has been incorrectly assigned. So the average number of matching elements after picking an incorrect bit is just $(1-\frac{5}{8})^{-1} = \frac{8}{3}$. This shows that a *single* power trace suffices to determine e completely except perhaps for the final two or three bits – and they are easily determined by comparing final outputs, which should equal $Y = A^e \bmod N \equiv 1 \otimes_N (A^e R)$.

We conclude that, as a matter of routine, any document digest should be combined with an unseen random component prior to signing. In particular [5], if v is random and d is the public key associated with e, then the attack is confounded by first replacing A with ARv^d, exponentiating as before, and then dividing by $v \bmod N$. However, such a blinding process fails to disrupt the attacks of §3.1 and §3.2 since they do not depend on knowledge of the inputs.

4 Computer Simulation

We built a computer simulation of 4-ary exponentiation for 384-bit exponents using 8-, 16- and 32-bit arithmetic and an implementation of Montgomery multiplication which included the final conditional modular adjustment which has been assumed throughout. The size of the arithmetic base made no difference to the results, as one can easily ascertain.

First, a random modulus and exponent were generated and fixed for the set of observations. Then a random input in the range $(0,N)$ was generated and scaled by R in the usual way, namely Montgomery-multiplying it by R^2. This first scaling enabled the observations to be partitioned according to whether or not an extra subtraction occurred. If X was the output from this, the next process computed and stored X^2 and X^3. The output data was partitioned according to whether or not subtractions were observed here too, giving 8 subsets in all. The exponentiation algorithm then repeatedly squared the running total twice and, according to the value of the next pair of exponent bits, multiplied in either X, X^2 or X^3. These three initial powers of X were always chosen as the "B" argument in the modular multiplication. The A input was the accumulating partial product and therefore the output from two successive squares. For each of the 8 subsets, the total number of extra subtractions were recorded for each multiplicative operation in the exponentiation.

As in Schindler [8], squares showed up clearly from multiplications by their lower number of subtractions when the full set of all observations (the union of the 8 subsets) was considered. To complete the determination of the exponent, it was necessary to establish which of X, X^2 or X^3 had been used in each multiplication. Already, a sequence of 4 successive squares indicated the positions of all the 00 bit pairs in the exponent. The partitioning into 8 subsets resulted in values for the B inputs which had different average properties. Consequently, for each subset, different frequencies of extra subtractions were observed. For multiplications with the same value of B the proportion of extra reductions in the subset tended to the same limit, but for those with different values of B, as expected, the limits were different. Selecting different subsets of the partition accentuated or diminished these differences. Combining the results from the best differentiated subsets, it was easy to determine which exponent bit pair had been used. Not only did the investigation enable the deduction of equivalence classes of equal digits, but knowledge of which subset was associated with which combination of subtractions in the initialisation process enabled the digits to be correctly assigned. Only the first one or two exponent digits were unclear, and this was because of the lack of independence between the arguments in the corresponding multiplications.

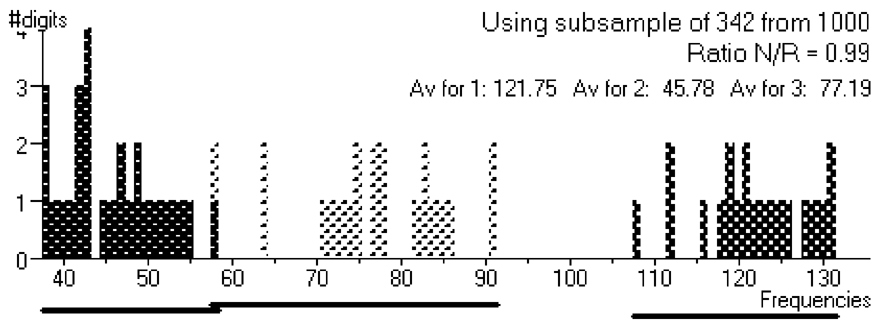

Fig. 2. Simulation: Set for Squares with Subtraction

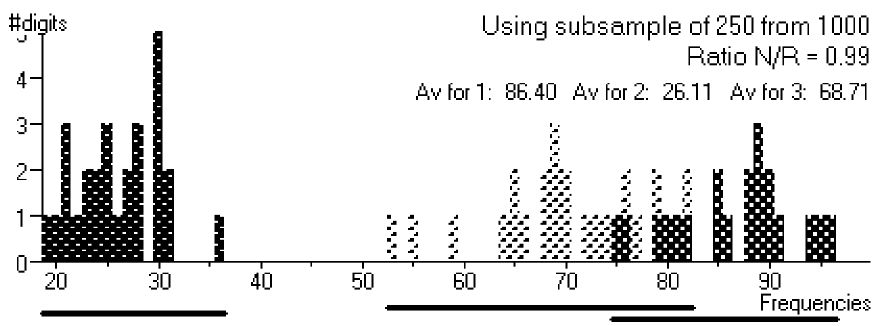

Fig. 3. Simulation: Subset for Cubes without Subtraction

It turned out that it was most helpful to look at the one third of observations for which the initial computation of X^2 generated an extra subtraction and partition this set according to whether the initial formation of X^3 had an extra subtraction or not. For both subsets, exponent digit $1 = 01_4$ generated the largest number of extra subtractions, digit $2 = 10_4$ the next largest and digit $3 = 11_4$ the smallest number. So a graph of digit positions marked along a frequency axis showed the digit positions clustering around three distinct sites. Switching between the two sets shifted the relative position of the digit $3 = 11_4$ instances in relation to the other two, making it possible to distinguish those digits from the others.

This is illustrated in Figures 2 and 3, where the three thick lines under the graph bracket together the positions of equal digits. The two sets enable an immediate, clear, correct association of digits with exponent positions. The illustrations are for 1000 samples when $N/R \approx 0.99$. Certainly, a smaller sample would have sufficed: half this number can almost be done by eye. For $N/R \approx 0.51$, about twice these sample sizes are required for the same degree of resolution. Notice that these sample sizes are independent of the key size because the probability of an extra subtraction is independent of the key size.

We did not attempt to perform a thorough analysis of the simulation output to see how few observations were necessary to guarantee that the correct exponent could be obtained. The digits which were most likely to be incorrectly assigned were those with subtraction frequencies furthest from the average for the digit. With sufficient observations to separate most of the non-zero digits into one of three classes, the potentially incorrect digits were clearly visible. Then, providing the number of such digits was small enough, every alternative could have been tested individually using other known data. Of course, in the presence of noisy readings, many more observations may need to be made, whilst if the data is clean enough, the results show that, in the absence of counter-measures, the safe lifetime of the key is too short for practical purposes.

5 Discussion

Any modular multiplication algorithm used in a smart card may suffer a problematic conditional subtraction of the type considered here in order to keep the result from overflowing. This is true not just for Montgomery modular multiplication but also for the classical algorithm, where the multiple of the modulus for subtraction is estimated from the top two or three digits of the inputs. Since the result is an approximation, a further conditional subtraction may be requested to obtain a least non-negative result. This subtraction is also open to attack in the above manner.

If the conditional modular reduction is performed every time and the previous value or new value is selected as appropriate, the movement of data may still betray whether or not the reduction is happening. Alternatively, an overflow bit can be stored and processed like another digit of the operand. This may cause exactly the timing variation that we should be trying to avoid. If not, then processing a top digit of 0 or 1 might still be easily recognised.

A general conclusion is therefore that N should be reduced away from a word boundary or register working length sufficiently for the modular multiplication algorithm to avoid any overflow to an extra word.

6 Counter-Measures & Conclusion

A detailed analysis has been presented showing how conditional subtractions at the end of Montgomery modular multiplications can be used very effectively to attack an RSA exponentiation with unknown modulus and secret exponent. The attack does not require knowledge of the plaintext input and can be applied successfully to the m-ary and sliding windows methods of exponentiation as well as to the standard square-and-multiply methods. Moreover, it applies in the same way to many other implementations of modular multiplication.

Computer simulations showed that if the data is clean enough to pick out each subtraction with high accuracy, then very few encryptions (under 1000) need to be observed before the exponent can be determined as a member of a small enough set for all possibilities to be tested individually. Furthermore, this number is independent of the key length.

There are simple counter-measures to avoid the problem. One of these is to modify the exponent by adding a random multiple of $\phi(N)$ before each exponentiation [5] so that the exponent digits are changed every time. This defeats the necessary averaging process over many observations which is the usual key to a successful side-channel attack.

Acknowledgment. The authors would like to thank A. V. Borovik who contributed to the key ideas presented here through a private communication [1].

References

1. A. V. Borovik & C. D. Walter, *A Side Channel Attack on Montgomery Multiplication*, private technical report, Datacard platform[7] seven, 24th July 1999.
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, 175–190.
3. D. E. Knuth, *The Art of Computer Programming*, vol. 2, Seminumerical Algorithms, 2nd edition, Addison-Wesley, 1981.
4. Ç. K. Koç, *High Radix and Bit Recoding Techniques for Modular Exponentiation*, International J. of Computer Mathematics, **40** (1991) no. 3-4, 139–156.
5. P. Kocher, *Timing attack on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Proc. Crypto 96 (N. Koblitz, ed.). Lecture Notes in Computer Science, **1109**, Springer-Verlag, 1996, 104–113.
6. P. L. Montgomery, *Modular multiplication without trial division*, Mathematics of Computation, **44** (1985), no. 170, 519–521.
7. R. L. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, **21** (1978), 120–126.
8. W. Schindler, *A Timing Attack against RSA with Chinese Remainder Theorem*, Cryptographic Hardware and Embedded Systems (CHES 2000), Christof Paar & Çetin Koç, editors, LNCS **1965**, Springer-Verlag, 2000, *to appear.*
9. C. D. Walter, *Montgomery Exponentiation Needs No Final Subtractions*, Electronics Letters, **35**, no. 21, October 1999, 1831–1832.
10. C. D. Walter, *An Overview of Montgomery's Multiplication Technique: How to make it Smaller and Faster*, Cryptographic Hardware and Embedded Systems (CHES '99), C. Paar & Ç. Koç, editors, LNCS **1717**, Springer-Verlag, 1999, 80–93.