Sliding Windows Succumbs to Big Mac Attack

Colin D. Walter

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

Abstract. Sliding Windows is a general technique for obtaining an efficient exponentiation scheme. Big Mac is a specific form of attack on a cryptosystem in which bits of a secret key can be deduced independently, or almost so, of the others. Here such an attack on an implementation of the RSA cryptosystem is described. It assumes digit-by-digit computations are performed sequentially on a single k-bit multiplier and uses information which leaks through differential power analysis (DPA). With sufficiently powerful monitoring equipment, only a small number of exponentiations, independent of the key length, is enough to reveal the secret exponent from unknown plaintext inputs. Since the technique may work for a *single* exponentiation, many blinding techniques currently under consideration may be rendered useless. This is particularly relevant to implementations with single processors where a digit multiplication cannot be masked by other simultaneous processing. Moreover, the *longer* the key length, the *easier* the attacks becomes.

Key words: Cryptography, RSA, differential power analysis, blinding, DPA, smart card, exponentiation, sliding windows.

1 Introduction

Timing analysis and differential power analysis (DPA) techniques [8], [9], [2], [1] show that RSA cryptosystems [13] suffer from implementation weaknesses rather than lack of algorithmic strength. The secret signing or decryption exponent d often seems easy to recover from a smart card or other dedicated embedded system using DPA [8], [1], [3], [10], [4]. These attacks start by averaging a number of power traces in order to remove dependencies other than the quantity being sought and to reduce the effect of random noise. For the card described in [10] which uses the standard square and multiply algorithm for exponentiation, this immediately reveals the exponent because of the different shape of power traces for squarings and multiplications.

The power-related property on which the current attack is based depends on the fact that switching a gate consumes more power than not doing so. Generally, these tiny effects are submerged in too many other data dependent variations to be easily extracted. However, here we develop a novel way of combining sections of power traces which enhances the effect into a potentially very powerful technique. We show that different multiplicands can be distinguished. As a result, the so-called m-ary and *sliding windows* methods of exponentiation [6], [7] become vulnerable as well as the square-and-multiply method.

A generally touted solution to this problem is to use different exponents with a randomly generated component on each decryption. In particular, Kocher, [8] §10, suggests using $d+r\phi(M)$ as the decryption key instead of d where M is the modulus and r is a random number generated anew for each decryption. This blinding certainly hides the exponent if averaging over a number of different decryptions has to be performed in order to reduce noise to levels at which the data dependencies are revealed. However, our simulations suggest that combining different sections of the power trace for just a *single* exponentiation may be sufficient to reveal the exponent, thereby negating the value of this type of blinding. Without such blinding, the technique certainly reduces the sample set that needs to be considered for DPA to be successful and implies that some sort of blinding should be a requirement in relevant cryptographic standards.

A *Big Mac Attack* on a secret key d is a method which enables d to be revealed bit by bit by nibbling at sections of d in any order. The implied independence of the derivation of different bits means that the total data and processing time required are only *linear* in the key length. This contrasts strongly with the *mathematical* strength of RSA, which is believed to be exponential in the key length. A well known brand product is so generously large as to be impossible to have a bite taken out of the whole at one go − like the method of attack, it must be nibbled at and consumed by tackling individual layers one by one in any order. Using DPA or other source of side-channel leakage, a similar arbitrary order of considering bits can eventually reveal the whole key, as we demonstrate. An example of another such attack, using timing information, was given in [16].

The context in which the attack may be mounted is a typical one for small embedded systems such as smart cards. We just require that a single k-bit multiplier be used to perform the RSA exponentiations in a digit sequential fashion, preferably with no other concurrent processing in progress.

2 Notation

An RSA cryptosystem (*resp.* signature scheme) over the integers [13] consists of a modulus $M = PQ$, which is the product of two large primes, and two keys d and e satisfying $A^{de} \equiv A \bmod M$. Message blocks A satisfying $0 \leq A < M$ are encrypted (*resp.* verified) with $C = A^e \bmod M$ and decrypted (*resp.* signed) using $A = C^d \bmod M$. The key e is generally chosen small with few non-zero bits (e.g. a Fermat prime, such as 3 or 17) so that encryption is relatively fast. The key d must be picked to satisfy $de \equiv 1 \bmod \phi(M)$ and therefore it usually has length comparable to M. The owner of the cryptosystem publishes M and e but keeps secret the factorization of M and the key d. Breaking the system means discovering d and is equivalent to factoring M, which is computationally infeasible for the size of primes used.

The computation of $A^d \bmod M$ is characterised by two main processes: modular multiplication and exponentiation. Our main assumption is that the implementation has a k-bit architecture and uses a single $k{\times}k$-bit multiplier to compute modular products $(A{\times}B) \bmod M$. So, except for the exponents d and e, each number X has a representation of the form $X = \sum_{i=0}^{n-1} x_i r^i$ where $r = 2^k$ is the *radix* or *base* of the representation, the coefficients x_i are its digits, and n is the number of digits required. The precise form or range of these digits is not important but we will see later that the larger n is, or the smaller k is, the more likely the attack is to succeed. The method is easily adapted to cases where the digit multiplier is not square.

2.1 Exponentiation

Exponentiation is often performed using the m-ary method [6] for which the exponent uses a representation with base m (here assumed to be a power of 2): $d = \sum_{i=0}^{t-1} d_i m^i$. The powers $C^i \bmod M$ $(i = 1,2,...,m{-}1)$ are pre-computed and allocated to table entries $C^{(i)}$. Then a partial product is repeatedly raised to the power m by squaring and the pre-computed power of C corresponding to the next digit of d multiplied in:

THE m-ARY (MODULAR) EXPONENTIATION ALGORITHM

{ Pre-condition: $d = \sum_{i=0}^{t-1} d_i m^i$ }
$\mathtt{C}^{(1)}$ `:= C ;`
`For i := 2 to m-1 do`
    $\mathtt{C}^{(i)}$ `:=` $\mathtt{C}^{(i-1)}{\times}\mathtt{C}$ `mod M ;`
`P :=` $\mathtt{C}^{(d_{t-1})}$ `;`
`For i := t-2 downto 0 do`
`Begin`
    `P :=` $\mathtt{P}^m$ `mod M ;`
    `If` $d_i \neq 0$ `then P :=` $\mathtt{P}{\times}\mathtt{C}^{(d_i)}$ `mod M ;`
`End ;`
{ Post-condition: $P = C^d \bmod M$ }

The *sliding window* technique [7] is a straightforward generalisation of this which makes more efficient use of the presence of zero bits in the exponent. It employs a mixed basis representation of the exponent, using powers of 2 and m. Only the odd powers $C^{(i)}$ need to be pre-computed and stored. The attack described here applies identically to this technique apart from the obvious modifications as a result of slightly different pre-computations, so it suffices to illustrate the ideas using the m-ary method.

Hardware power consumption depends critically on bus movement involved in low level operations such as fetching instructions, reading from and writing to memory, etc. Since the long integer multiplications take a large number of cycles to perform and a large number of consecutive multiplications are executed, attackers are usually able to establish correctly the boundaries in the power traces between the operations in the algorithm above.

2.2 Modular Multiplication

Each long integer multiplication or squaring consists of a large number of individual digit-by-digit multiplications. Normally the modular reductions are interleaved within the iterations of the multiplication:

Classical Modular Multiplication Algorithm:

{ Pre-condition: $A = \sum_{i=0}^{n-1} a_i r^i$ }

```
R := 0 ;
For i := n-1 downto 0 do
Begin
     R := r×R + a_i×B ;
     q_i := R div M ;
     R := R - q_i×M ;
End ;
```

{ Post-condition: $R \equiv (A\times B) \bmod M$ }

Montgomery's version of long integer modular multiplication [11] has a similar structure, just reversing the order of processing the digits a_i.

Both the classical algorithm above and Montgomery's version are usually implemented in a way which makes them behave identically as far as this attack is concerned. The main variation worth highlighting is that for each long integer multiplication of the exponentiation either input A or B may be chosen as the pre-computed power of the initial ciphertext C. For convenience, we assume this power of C is the first argument, namely A, in the above code. However, to avoid unnecessary movement of data, the hardware must usually choose the same order for every multiplication. Then it is easy for an attacker to try both possibilities and select the one which provides the expected correlations.

3 Selecting & Averaging the Power Traces for Big Mac

The attack requires side channel leakage which has a dependency on the data being processed by the multiplier. Apart from measuring power consumption of the whole chip [4], the methods of Gandolfi *et al.* [5] could be directed to measuring EMR from the multiplier itself.

Assume that discrete sampling of the cryptographic device provides a power (or EMR) trace function $tr : \mathbf{Z} \to \mathbf{R}$ for the pre-computations and exponentiation for a single decryption or signing. The definition of tr outside this computation interval is irrelevant here. Suppose further that the regular sampling provides a non-zero number of values for every digit multiplication. The more frequent the sampling, the better the results obtained for this attack, especially if a number of measurements can be made during each clock cycle. Typically, the standard smart card clock runs at 3.57 MHz and the current is sampled at 200 MHz, yielding a ratio of nearly 2^6 to 1. This current is recorded using one or two bytes per measurement. As far as possible, such sampling should be synchronised to take place at the same points of each clock cycle.

Sommer [12] noted that certain points in the clock cycle have much greater value for determining data dependencies than others. Initially, as gates are switched along paths in the multiplier, the current will be higher and be dependent on the activity. However, at the end of a clock cycle the combinational logic should have stabilised, and it will have a much lower data dependent contribution. We are only interested in points with data dependent power consumption. Assume that several such points have been identified in the clock cycle, and we are able to take a weighted average of them in the trace so that, as far as possible, the data dependent contribution to the power represents the number of gates being switched and any measurement errors are averaged out. All other points must be discarded from the trace, leaving only data dependent ones.

The main loop of the long integer modular multiplication algorithm contains a repetition of k-bit multiply-accumulate digit operations of the form

$$r_j + r\times carry \;\; := \;\; r_{j-1} + a_i\times b_j + carry \qquad (0 \leq j < n)$$

which take place in a single cycle. It is only the sub-traces for these operations that are used in the attack. The sections of the trace corresponding to these can be identified easily because, by using the multiplier, they differ substantially from sections corresponding to other operations.

Suppose we have already distinguished squares from multiplies and wish to establish the value of the exponent digit, say d_s, associated with the sth long integer multiplication. Let tr_{sij} denote the function obtained by setting tr to 0 outside the sub-interval during which the attacker expects the digit product $a_i\times b_j$ to be computed within the sth multiplication, and then translating that subinterval to $[i\tau,\ (i+1)\tau-1]$ where τ is the common number of sample points for each such digit-by-digit multiply-accumulate. (After deleting irrelevant points and averaging as necessary, we may well have reduced τ to 1.)

Assuming, as stated, that A is the input which is a pre-computed power of C, define $tr_{si} = \frac{1}{n}\sum_{j=0}^{n-1} tr_{sij}$ to be the function given by averaging the tr_{sij} over all j. So tr_{si} depends on the single digit a_i of A but all the digits b_j and r_j of essentially random numbers B and R, and some carries. This averaging should produce a function tr_{si} for which the random variable associated with the value at any given point has contributions to the variance from its dependence on B and from random noise, both of which are only $\frac{1}{n}$ times those for a_i and for equivalent positions in tr. Because the multiply-accumulate operation uses k times as much hardware in a_i- and b_j-dependent computations than for accumulating the *carry* and r_{j-1} digits, the contributions from r_{j-1} and the *carry* are certainly lower, perhaps by k times, than that from B. Hence the clearest correlation that tr_{si} should exhibit will be with the value of a_i.

This averaging of the traces over the digits of B replaces the usual DPA averaging of traces over a number of different exponentiations. On the reasonable assumption that B is sufficiently random and has a number of digits, the resulting average trace will then have little dependence on B. (If the pre-computed power is the B input, we sum over i instead of j to obtain a result which again depends on a single digit of the pre-computed power of C.)

Lastly, define $tr_s : \mathbf{Z} \rightarrow \mathbf{R}$ by $tr_s = \sum_{i=0}^{n-1} tr_{si}$. As tr_s is the concatenation of the non-zero sections of the tr_{si}, it has a non-zero definition on $[0, n\tau-1]$ whose strongest data dependency is through the pre-computed argument A of the sth multiplication, i.e. the power of C corresponding to the exponent digit d_s. The obvious question to ask is whether this dependency is strong enough to identify d_s since then the secret exponent d can be discovered.

4 Simulation

In order to investigate the feasibility of the attack, a simple k-bit multiplier was simulated. It was built mostly from standard 3-to-2 full adders with a carry propagator and had a variable size k. This was used to count gate switching in the combinational logic only, with no account being taken of changes in registers which might contribute to power use.

Data-dependent power usage is immediately apparent when gate counts are partitioned into subsets according to the Hamming weight of the two inputs. There is a very clear increase in the number of gate switchings as the Hamming weight of either input is increased. Tables of these values displayed a difference of a little over k gate changes between adjacent cells in the centre of the table, where both Hamming weights are approximately $k/2$ and most input pairs are clustered. Except for extreme Hamming weights, the table entries were almost linear in each Hamming weight − sufficiently so to explain and justify the arguments below. Moreover, the results were essentially symmetrical, i.e. the same number of gates were switched on average when the two inputs were interchanged. This occurred under several configurations even though no attempt was made to balance the number of gates switched.

For a variety of values of k, modulus bit lengths and exponent bases m, a number of random sets of powers $\{C^{(1)}, C^{(2)}, ..., C^{(m-1)}\}$ were generated. These were used as input A of the modular multiplier and, to simulate the pre-comuptations, multiplied by a random long integer B to create a trace tr_i associated with each $C^{(i)}$. The trace consisted of a vector of gate switch counts for each digit of $C^{(i)}$. These individual counts were the sum of the gate switch counts for each product of the digit of $C^{(i)}$ by a digit of B. The traces then corresponded to the power traces tr_s. With the component from B averaged, the trace tr_i for each $C^{(i)}$ corresponded closely to the vector of true average gate switch counts for the digits of $C^{(i)}$. In particular, this meant the trace was reasonably characteristic of $C^{(i)}$ and its elements were closely related to the Hamming weights of the digits.

To simulate exponentiation multiplications, another random long integer B' was chosen, multiplied by a random member of $\{C^{(1)}, C^{(2)}, ..., C^{(m-1)}\}$, and the trace $tr_{B'}$ of gate switch counts created. Like tr_B, it was close to the true average gate switch counts for whichever $C^{(i)}$ had been selected. The trace was matched up with the traces tr_i of each $C^{(i)}$. Specifically, the Euclidean distance between it and every tr_i was computed, and the closest chosen to predict i.

Multiplier Size	$k = 64$	$k = 32$	$k = 24$	$k = 16$	$k = 8$
Av to nearest	4973	2709	2538	2428	2245
SD to nearest	2582	1482	1334	1183	1024
Av to others	17981	24312	19834	23475	19793
SD to others	1232	513	408	481	217

Table 1 Gate Switch Statistics for 512-bit Modulus with $m = 4$.

The attack simply requires this prediction to be correct. For many typical values of k, n and m, the attack invariably succeeded. Table 1 gives the means and standard deviations for i) the distances between $tr_{B'}$ and the correct tr_i and ii) $tr_{B'}$ and the incorrect tr_i. The difference between the two cases is startlingly large. Table 2 shows low error frequencies even in the worst cases, namely for the largest k and smallest n. If the number of bits in the modulus length is fixed, then the average distance to the nearest trace increases as k increases so that difference between nearest and non-nearest traces decreases. For fixed k, increasing the size of the modulus provides more digits over which to average and more elements in the vector, thereby improving the ability to determine the multiplier correctly. As one would expect, increasing m just makes the nearest trace closer and increases the variance in the distances to the rest.

Modulus Length	256 bits	384 bits	512 bits	768 bits	1024 bits
Av to nearest	1529	2366	3750	4501	6246
SD to nearest	885	1403	2386	2535	3612
Av to others	5890	11753	17896	32594	53070
SD to others	1108	2412	2279	4646	4581
%age errors	0.9284	0.1155	0.2819	0.0000	0.0000

Table 2 Gate Switch Statistics for 32-bit multiplier with $m = 8$.

Squares and random products were distinguishable from multiplications by a $C^{(i)}$ because their traces were not close to any tr_i. Indeed, the statistics for each were similar to the non-nearest table entries. Thus, all long integer multiplicative operations, including squares, could normally be correctly distinguished in the simulation and hence the secret key recovered.

5 Distances between Power Traces

Suppose tr_{s1} and tr_{s2} are a pair of power traces constructed as above for the $s1$th and $s2$th multiplications of the exponentiation. As the traces are real-valued functions on the integer subinterval $[0,\ n\tau{-}1]$, they represent points in $\mathbf{R}^{n\tau}$. Define d to be the Euclidean metric on $\mathbf{R}^{n\tau}$ and let $d(s1, s2)$ be the distance between the points defined by tr_{s1} and tr_{s2}. One advantage of such a metric is that places where the traces differ most contribute much more highly to the distance between traces than places with the smallest differences. This should help to emphasise the contribution from parameter A, which is approximately n times the contribution from other parameters. It is important to omit from this metric the points without a noticable data-dependent contribution as they reduce the visibility of the data dependence which needs to be observed.

For equal exponent digits $d_{s1} = d_{s2}$ the corresponding multiplications share the same first argument. Since there are no other strong data dependencies, the value of $d(s1, s2)$ should be small, corresponding purely to noise and variation from the average of the digits appearing in the other arguments. For different exponent digits $d_{s1} \neq d_{s2}$ the value of $d(s1, s2)$ should be noticeably larger because of the greater dependence on the first arguments, which are different.

According to the simulation, the data dependent contribution to power consumption is roughly proportional to the Hamming weight of the arguments. So we can expect the distance between two traces to be approximately related to the distance between the vectors consisting of the Hamming weights of the digits of the multipliers A. Since the Hamming weights of digits are distributed binomially, it is easy to obtain statistics for the random variable associated with the distance between two such vectors and see that it has very similar behaviour to that observed in the simulation. Hence this gives an accurate guide to the effect of changing any parameters and enables accurate error predictions to be made. In particular, it justifies the observation that distances between pair of traces cluster around two points, one of which is 0.

6 Identifying Equal Exponent Digits

Next we present an algorithm for partitioning the set $T = \{0, 1, 2, ..., t{-}1\}$ of exponent digit indices into subsets for which the corresponding digits of d are the same. This partition, $\wp$, has to define m subsets, one for each (exponent) digit value in base m. The subset containing the zero exponent digits should already have been identified by using the ability to distinguish between (long integer) squares and multiplies to observe which exponent digits have no corresponding muliplication in the exponentiation algorithm. For the other digit subsets, the association of each subset with a particular non-zero base-m digit is performed in the next section.

The algorithm puts the indices either into a new subset of the partition, or into the subset of indices which is "nearest" in an obvious sense: the distance between a single point s and a non-empty set of points S is defined here as $d(s, \bar{S})$ where $\bar{S}$ is the centroid of S, i.e. $\bar{S} = |S|^{-1} \sum_{s' \in S} s'$.

For each pair of (non-zero) exponentiation digits with indices $s1$ and $s2$, arrange the distances $d(s1, s2)$ into descending order, and set up $m{-}1$ buckets to receive sets of indices, one for each exponent digit value. Then consider the pairs

$(s1, s2)$ in order of decreasing distance between their two traces:

i) If both indices are in different buckets, then move to the next pair.
ii) If there is one unassociated index and an empty bucket then place that index in the empty bucket and again move on to the next pair.
iii) If neither index is associated with a bucket and there are (at least) two empty buckets, put the indices into separate empty buckets, and move to the next pair.
iv) If both indices are in the same bucket, compute the distances of $s1$ and $s2$ from the set of indices in each bucket. If both are already in the nearest bucket, move on to the next pair, but otherwise, move $s1$ and $s2$ into their nearest buckets, moving the nearer one first and recomputing distances before moving the second. Then move to the next pair.
v) If there is an unassociated index and no empty bucket, then put the new index in a temporary extra bucket, compute the distances between every pair of buckets and combine the pair of buckets which are the shortest distance apart to restore the original number of buckets. Move to the next pair.
vi) If neither index is associated but there is only one empty bucket, compute the distances from $s1$ and $s2$ to each non-empty bucket. If $s1$ is the nearer to its nearest bucket, then put $s1$ into that bucket and $s2$ into the remaining empty bucket. Otherwise, put $s2$ into its nearest bucket and $s1$ into the empty bucket. Then move on to the next pair.
vii) If neither index is associated and there are no empty buckets, then perform (v) for both $s1$ and $s2$ individually.

With perfect data, the algorithm should first treat all the pairs $(s1, s2)$ which correspond to different exponent digits and correctly put them into different buckets or find that they are already in different buckets. Then, from some point on, all pairs correspond to equal digits and so the indices should be found in the same bucket. The algorithm does not place indices in the same bucket until there are no empty buckets left. So it is likely for indices with the same exponent digit to be initially spread over several buckets. These buckets then need to be coalesced to provide empty buckets for unassociated indices. Process (v) does this. Once there are no empty buckets left, then action (iv) is used to ensure that the best assignments have been made previously.

With perfect information, each element of T can be assigned to one of the partition subsets by calculating at most $m{-}1$ distances. So fewer than mt distances are required to establish the partition correctly if all distances are clearly and correctly distinguished as small or not. Hence, with up to $t(t{-}1)/2$ pairs in total, there is considerable extra information to improve and confirm the construction of $\wp$ as it progresses. However, in case of error, all assignments can be ranked using distances to buckets, and the most likely tried first for correctness.

7 Associating Digit Values with Exponent Positions

The partition $\wp$ yields $(m-1)!$ possibilities for the key d, corresponding to the possible associations[1] of non-zero digits from 1 to $m-1$ with the $m-1$ different non-zero equivalence classes for the induced equivalence relation on $T = \{0, 1, 2, ..., t-1\}$. However, the pre-computation of the powers $C^{(i)}$ for $i = 1, 2, ..., m-1$ means that we have a known multiplication involving $C^{(i)}$ for each exponent digit except 0 and $m-1$, namely $C^{(i+1)} = C^{(i)} \times C \bmod M$ in the case of m-ary exponentiation and $C^{(i+2)} = C^{(i)} \times C^{(2)} \bmod M$ in the case of sliding windows.

Following the algorithm of the previous section, each trace tr_i corresponding to the pre-computational multiplication with first argument $A = C^{(i)}$ is associated with its nearest bucket of exponent digit indices. This bucket is then labelled "i" and should correspond to exponent digit i. Ideally, this should not associate two labels with one bucket, and should leave one bucket unlabelled. This last bucket is labelled with the remaining exponent digit, namely $m-1$.

If inconsistencies arise from this labelling, then it is easy to rank each possible labelling using distances from each tr_i to each bucket. Each labelling can be tried in turn until overall consistency in achieved. As the m-ary method uses significant memory when used in an embedded cryptographic device, m is usually very small. So all $(m-1)!$ possibilities could be tested for correctness if necessary.

The trace-averaging process depends on the randomness of the B input and its independence from the A input in order to obtain a result which characterises the A input. During the pre-computations, both inputs are powers of the initial text C and therefore not independent of each other. However, since 3 is generally regarded as an acceptable encryption exponent, we can assume that the powers $C^{(i)}$ are sufficiently independent of C when i contains an odd divisor. Then the traces tr_i should be acceptable for every i which is not a power of 2. Assuming also that problems with powers of 2 decrease as the power increases, only traces for the exponent digits 1 and 2 might display dependency problems.

For digit 1, the power trace for $C^{(2)} = C \times C \bmod M$ depends on both arguments. We present two solutions to this. First, one can expect to identify which subtraces corresponding to the digit products $a \times a$. They can be excluded from the averaged trace to obtain a new trace which at each point depends on a single digit of C and some other effectively independent, random digits. Such a revised trace behaves like the other averaged trace functions. Alternatively, we may assume $m > 2$ since if $m = 2$ there is nothing to decide: all the non-zero exponent digits must be 1. Each product $C^{(i+1)} = C^{(i)} \times C \bmod M$ involves C as the *second* argument rather than the first. Thus, for any one of these multiplications, one can average the traces in a different way, this time summing over the different first digits while the second is kept fixed, rather than vice versa. Then

[1] We have not assumed any knowledge of the modulus M. However, as Adi Shamir pointed out during the presentation, if M and e *are* known, then in this section one can probably make the correct association by using the fact that the bits of the top half of the exponent coincide with those of a small multiple of M.

for $m > 2$ the last such multiplication gives an alternative to the initial squaring used in the first method for providing a trace for C. If $m > 4$ then the remarks above about 3 as an encryption exponent establish that the two arguments are effectively independent when the last multiplication is used for a trace for C. However, if $m = 4$ then this multiplication is the product of C and $C^{(2)}$ and there may be cause for concern. We remark on this potential problem next, but otherwise it is reasonable to assume that a typical trace can be obtained for the class of the exponent digit 1 from the pre-computation multiplications.

The trace associated with digit 2 is derived from the product of $C^{(2)}$ and C. Using the second alternative above, this may also be the source of the trace associated with digit 1. However, the dependence between these arguments should be very weak since an essentially random multiple of M has been subtracted from C^2 to obtain $C^{(2)}$. So a usable trace should also be obtained for digit 2.

8 Big Mac

By omitting the cross-checking afforded by comparing multiplications of the exponentiation, we obtain the Big Mac attack in which exponent digits are determined independently, as in the simulation section. Each trace tr_s from a multiplication in the exponentiation is compared with each trace tr_i from the pre-computations and the nearest is selected to determine the exponent digit at position s. When no pre-computation trace is close to tr_s then digit $m-1$ (for which there is no pre-computation trace) is assigned. All t exponent digits can then be recovered in t times the time required for recovering one digit. Moreover, apart from pre-computations, only the power trace for a single multiplication is used to recover a single exponent digit. So t times the data, i.e. the whole exponentiation record, is required to recover all digits.

More precisely, suppose k and m are fixed and, as usual, $t \approx nk/\log_2 m$. We are interested in what happens when the bit length nk of the arguments is varied. For each long integer multiplication the number of k-bit multiplications is $O(t^2)$. But, for a common level of accuracy, all averaged traces could be compiled from a *fixed* number of these digit-by-digit multiplications which is independent of t. This would use only constant data per exponent digit and consequently $O(t)$ data for the whole attack. If the full quantity of data is used, the traces tr_s become more accurate as t (or nk) increases. Furthermore, if every pair $(s1, s2)$ is considered, then more cross-checking is possible as t increases. Hence, the attack becomes much more viable for larger keys!

9 Using a Set of Exponentiations

The method of attack described so far has been developed from the power trace associated with a single exponentiation. It depended on a reasonable separation between the powers of the initial input C when measured using the Euclidean metric on the associated vectors of digit Hamming weights. If any powers of C are too closely related the attack may fail to work. However, one could wait patiently for an input C where the Hamming weights of the pre-computed powers are sufficiently widely spread. For large n with small m, this should not take long.

To benefit from traces from a set of exponentiations, it is important *not* to average the traces. Instead, if the exponent is the same in each case, the sub-traces for each multiplication need to be *concatenated* to provide longer vectors for comparison. Alternatively, an observation matrix can be constructed with a row for each exponentiation and a column for each exponent digit index, and containing the best estimate for the exponent digit. Repeated use of the same digits at the same exponentiation points then leads to corresponding correlations between columns of this matrix. Standard statistical techniques should then reveal the exponent.

10 Some Final Details

10.1 Separating Squares and Multiplies

Finally, we consider some detail which, for the sake of simplicity, was left out of the above arguments. The first concerns differentiating squares from multiplies. The simulation section noted that squares behaved like multiplications by $C^{(m-1)}$, having no nearest multiplier. Therefore using distances from pre-multiplication traces to classify all long integer operations will place both these types in the same bucket. Since each multiplication must be preceded and followed by r squarings, the determination of which is which should be straightforward. Moreover, the multiplications by $C^{(m-1)}$ should all be close to each other, whilst the squarings should not. Indeed, this also enables the case $m = 2$ to be cracked. Thus, if the attack separates the different multipliers, it certainly also separates the squares.

10.2 Initial Exponent Digits

The next omission relates to the initial few multiplications of the exponentiation after any pre-computation has taken place. The first value assigned to P in the exponentiation algorithm of §2.1 corresponds to the first (non-zero) digit of d and involves no multiplication. Hence the method here appears to yield no information about it. Thus there may be $m{-}1$ times more possibilities for d than estimated above, one for each choice of the first non-zero digit of d. This is followed by r squarings. The first is of $C^{(d_1)}$. However, a trace for $C^{(d_1)}$ can be extracted in the same way as described in §7 for obtaining a trace of C from computing $C^{(2)}$. This should reveal d_1 using the usual nearest bucket method. Once the multiplications for P do start, the B argument of the modular multiplication is generally no longer sufficiently closely related to influence the power trace adversely. The attack will therefore work successfully from this point on. The only noticeable exception is the first multiplication (as opposed to a squaring) when $m = 2$ and the second digit of d is 1.

10.3 Zero Multiplier Digits

The last concern is if zero digits (base r) occur in the inputs to a modular multiplication and optimization causes the associated digit multiplications to be skipped. To avoid timing attacks, this should probably not occur. However, with typical values such as $r \approx 2^{32}$, $n \approx 2^5$, $m = 4$ and $t \approx 2^8$ for 1024-bit keys, we have about $mn = 2^7$ digits among the pre-computed powers, and about $nt(m-1)/m = 1.5\times 2^{12}$ digits among the arguments B of the multiplications during an exponentiation. So the chances of encountering a digit 0 are small ($\approx nt/r$). In the unlikely event of a zero, the analysis should become much easier. If the zero digit lies in a pre-computed power, timing analysis immediately reveals which multiplications use that power. Otherwise the zero digit occurs in the B argument of a multiplication and one simply defines tr_{si} by averaging the traces over the non-zero digits of B. At worst, another decryption trace might be obtained to avoid the problem altogether.

10.4 Chinese Remainder Theorem

Implementations using the Chinese Remainder Theorem can be attacked in the same way because having a single digit multiplier forces the two exponentiations to be performed sequentially. The two exponents are then recovered one after the other in the way described above, yielding the secret key.

11 Conclusion

An unknown plaintext DPA attack on a single RSA exponentiation has been described where the implementation uses a single k-bit multiplier. This may well prove successful, particularly against a RISC processor where no other operations can be carried out to mask the multiplier's use of power. The attack becomes easier to perform accurately as the key length is increased because more useful data is available. For fixed k and using all available data, the running time is proportional to the key length cubed.

The attacker waits for a sufficiently helpful exponentiation, and then uses a careful and novel selection and combination of sections from a single power trace to recover secret decryption keys. If the same exponent is reused the attack becomes easier. Blinding keys is no defence if the attack succeeds on a single exponentiation. Then other methods are required. One solution might be to keep a processor/co-processor architecture where the two processes mask each other. Alternatively, a pipelined k-bit multiplier with several stages might be used, or CRT performed with the exponentiations using two separate multipliers in parallel. Yet another solution might be to use a systolic modular multiplier [15] where many unrelated digit multiplications are computed in parallel.

Certainly one concludes that performing a single, digit-level operation at one time, such as a multiplication, leads to a potentially unsafe implementation of the RSA cryptosystem.

References

1. D. Boneh, *Twenty Years of Attacks on the RSA Cryptosystem*, Notices of the AMS, **46**, no. 2, Feb 1999, pp 203-213.
2. D. Boneh, R. DeMillo & R. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults*, Eurocrypt '97, Lecture Notes in Computer Science **1233**, Springer-Verlag, 1997, pp. 37-51.
3. D. Chaum, *Blind Signatures for Untraceable Payments*, Proc. Advances in Cryptology (Crypto '82), Plenum Press, 1983, pp. 199-203.
4. J.-S. Coron, *Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems*, Cryptographic Hardware and Embedded Systems (Proc CHES 99), C. Paar & Ç. Koç editors, Lecture Notes in Computer Science **1717**, Springer-Verlag, 1999, pp. 292-302.
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 (*this volume*), Springer-Verlag, 2001.
6. D. E. Knuth, The Art of Computer Programming, vol. 2, *Seminumerical Algorithms*, 2nd Edition, Addison-Wesley, 1981, pp. 441-466.
7. Ç. K. Koç, *Analysis of Sliding Window Techniques for Exponentiation*, Computers and Mathematics with Applications, **30**, no. 10, 1995, pp.17-24.
8. P. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, Advances in Cryptology, Proc Crypto 96, Lecture Notes in Computer Science **1109**, N. Koblitz editor, Springer-Verlag, 1996, pp 104-113.
9. P. Kocher, J. Jaffe & B. Jun, *Differential Power Analysis*, Advances in Cryptology – Crypto '99, Lecture Notes in Computer Science **1666**, M. Wiener (editor), Springer-Verlag, 1999, pp 388-397.
10. 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.
11. P. L. Montgomery, *Modular Multiplication without Trial Division*, Math. Computation, **44**, 1985, pp. 519-521.
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. R. L. Rivest, A. Shamir & L. Adleman, *A Method for obtaining Digital Signatures and Public-Key Cryptosystems*, Comm. ACM, **21**, 1978, pp. 120-126.
14. 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.
15. C. D. Walter, *Systolic Modular Multiplication*, IEEE Transactions on Computers, **42**, no. 3, March 1993, pp. 376-378.
16. 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.