More Detail for a Combined Timing and Power Attack against Implementations of RSA

Werner Schindler[1] and Colin D. Walter[2]

[1] Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185-189, 53175 Bonn, Germany
Werner.Schindler@bsi.bund.de
[2] Comodo Research Laboratory
10 Hey Street, Bradford, BD7 1DQ, UK
Colin.Walter@comodogroup.com

Abstract. Implementations of Montgomery's modular multiplication algorithm (MMM) typically make conditional subtractions in order to keep the output within register or modulus bounds. For some standard exponentiation algorithms such as m-ary, it has been shown that this yields enough information to deduce the value of the exponent. This has serious implications for revealing the secret key in cryptographic applications without adequate counter-measures. Much more detail is provided here about the distribution of output values from MMM when the output is only reduced to keep it within register bounds, about how implementations of sliding windows can be attacked, and about handling errors.

Key Words. RSA cryptosystem, side channel leakage, power analysis, timing attack, Montgomery modular multiplication, exponentiation, statistical decision problem.

1 Introduction

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 [4], and the latter from data causing gate switching in the circuitry. Both manifest themselves measurably through overall current variation as well as local or global electro-magnetic radiation (EMR) [5–7]. Information leaking in these ways might be used by an attacker to deduce secret cryptographic keys which are contained securely inside a smart card.

Timing variation can be measured using overall delay during decryption or signing [1, 4, 11, 13]. However, for successful attacks on keys with limited lifespans for which only a bounded number of decryptions with the same key is allowed, some finer detail may become necessary, such as the time variations for executing component procedures. This detail can be seen by observing delays between the characteristic power consumption wave form for loading instructions and data at the start of the procedure. Here this is applied to the version of Montgomery modular multiplication [9] which contains a final conditional subtraction for reducing the output by only enough to fit within register bounds. The aim is to recover a secret RSA [10] exponent from deductions of whether or not this conditional subtraction occurs.

An essential assumption is that there is no blinding of the exponent or randomisation in the exponentiation algorithm so that the same type of multiplicative operations are performed (but with different data) every time the key is used. This enables an attacker to capture the collection of instances of the extra final subtraction for each individual operation. We will show how to determine exponent digits from this data, recover from any errors, and hence obtain the secret key even if the input text (the base of the exponentiation) has been blinded.

The results emphasise the need for care in the implementation of RSA. Indeed, similar results apply to elliptic curve cryptography (ECC) [8] and other crypto-systems based on modular arithmetic. In all cases where a key is re-used, some counter-measures should be employed to ensure that the conditional subtraction is removed, or at least hidden, and that the same sequence of key-dependent operations is not re-used for each exponentiation. In elliptic curve cryptography, standard key blinding adds about 20% to the cost of a point multiplication. The temptation to avoid this overhead should be resisted.

Montgomery modular multiplication [9] is arguably the preferred method for hardware implementation. We treat this algorithm here, but equivalent results should hold for any method for which there are correlations between the data and the multiplication time. The generic timing attack methodology which we adopt was first described in [16]. This approach established the feasibility of recovering the key within its lifetime and also gave the theoretical justification behind the different frequencies of final subtractions observed between squarings and multiplications.

A closer study of such distributions reduces the number of exponentiations required to recover the key. This was done for various settings in [11–13] when the final reduction yields output less than the modulus. However, reducing output only as far as the same length as the modulus is more efficient, and is the case studied here. The mathematics is more involved and so requires numerical approximation methods, but this is easy for an attacker. In addition, more detail is given concerning the recovery of the key when sliding windows exponentiation is employed.

2 The Computational Model

In order to clarify the precise conditions of the attack, each assumption is numbered. First, the computational model requires several assumptions, namely:

i) secret keys are unblinded in each exponentiation;
ii) Montgomery Modular Multiplication (MMM) is used with a conditional subtraction which reduces the output to the word length of the modulus;
iii) the most significant bit of the modulus lies on a word boundary; and
iv) exponentiation is performed using the m-ary sliding windows method [2, 3].

Exponent blinding might be deemed an unnecessary expense if other countermeasures are in place. So we still need better knowledge of how much data leaks out under the first assumption. For a sample of one exponentiation this particular assumption is irrelevant, but then, with data leaked from other sources, the techniques here might be just sufficient to make a search for the key computationally feasible.

As shown below, efficiency is good justification for the second hypothesis. The word length referred to there is the size of inputs to the hardware multiplier, typically 8-, 16- or 32-bit. However, standards tend to specify key lengths which are multiples of a large power of 2, such as 768, 1024, 1536 and 2048 bits. So the third hypothesis is almost certainly the case.

Because of its small memory requirements and greater safety if an attacker can distinguish squares from multiplications, a sliding window version of 4-ary exponentiation is the usual algorithm employed in smartcards. The exponent is re-coded from least to most significant bit. When a bit 0 occurs, it is re-coded as digit 0 with base 2 and when a bit 1 occurs, this bit and the next one are recoded as digit 1 or 3 with base 4 in the obvious way. This representation is then processed left to right, with a single modular squaring of the accumulating result for digit 0, and two modular squarings plus a modular multiplication for digits 1 and 3. According to the digit, the first or pre-computed third power of the initial input is used as the other operand in the multiplications.

An attacker can recover the secret exponent if he can i) distinguish squarings from multiplications, and ii) determine whether the first or third power was used as an operand in the multiplications. The classical square-and-multiply or binary algorithm ($m = 2$) is less secure against this type of attack as the second task is omitted. The attack here treats any m, and applies equally well when the Chinese Remainder Theorem (CRT) is used.

3 Initial Notation

Let R be the so-called "Montgomery factor" associated with an n-bit RSA modulus M. Then the implementation of Montgomery's algorithm satisfies the following specification [15]:

v) For inputs $0 \le A,\ B < R$, MMM generates output $P \equiv A{*}B{*}R^{-1} \bmod M$ satisfying $ABR^{-1} \le P < M{+}ABR^{-1}$ before any final, conditional subtraction.

Clearly this specifies P uniquely. The division by R is the result of shifting the accumulating product down by the multiplier word length for a number of iterations equal to the number of words in A. Since A, B and M will all have the same word length for the applications here, hypothesis (iii) yields

vi) $M < R < 2M$

i.e. R is the smallest power of 2 which exceeds M.

The output bound $P < M{+}R$ implied by (v) means that any overflow into the next word above the most significant of M has value at most 1. When this overflow bit is 1, the conditional subtraction can be called to reduce the output to less than R without fear of a negative result. Hence that bit can be used efficiently to produce an output which satisfies the pre-conditions of (v) for further applications of MMM. This is the case of MMM that is considered here.

Alternatively, the condition $P < M$ is often used to control the final subtraction. This is the standard version, but the test requires evaluating the sign of $P-M$, which is more expensive than just using the overflow bit. This case of MMM was discussed in [12]. A constant time version of MMM is possible by computing and storing both P and $P-M$, and then selecting one or other according to the sign of the latter. However, the subtraction can be avoided entirely if the inputs satisfy $A, B < 2M$ and R satisfies $4M < R$. This, too, is more expensive [15].

As usual, the private and public exponents are denoted d and e. For deciphering (or signing), ciphertext C is converted to plaintext $C^d \bmod M$. The m-ary sliding windows exponentiation algorithm using MMM requires pre-computation of a table containing $C^{(i)} \equiv C^i R \bmod M$ for each odd i with $1 \leq i < m$. This is done using MMM to form $C^{(1)}$ from C and $R^2 \bmod M$, $C^{(2)}$ from $C^{(1)}$, and then iteratively $C^{(i+2)}$ from $C^{(i)}$ and $C^{(2)}$. By our assumptions, the word length of each $C^{(i)}$ is the same as that of M, but it may not be the least non-negative residue. We also define $b = \log_2 m$ and assume it to be integral.

4 The Threat Model

The security threat model is simply that:

vii) an attacker can observe and record every occurrence of the MMM conditional subtraction over a number of exponentiations with the same key.

Section 1 provided the justification for this. The attack still works, although less efficiently, if manufacturers' counter-measures reduce certainty about the occurrence or not of the subtraction.

Unlike many attacks in the past, we make no assumptions about the attacker having control or knowledge of input to, or output from, the exponentiation. Although he may have access to the ciphertext input or plaintext output of a decryption, random nonces and masking should be employed to obscure such knowledge so that he is unable to use occurrences of the conditional subtraction to determine whether a square or multiply occurred. It is therefore assumed that

viii) the attacker can neither choose the input nor read either input or output.

Indeed, even without masking this is bound to be the case for exponentiations when the Chinese Remainder Theorem has been used. Lastly, as the attacker may have legitimate access to the public key $\{M, e\}$, it is assumed that

ix) the correctness of a proposed value for the private exponent d can be checked.

The assumptions so far mean that the same sequence of multiplicative operations is carried out for every decryption. So, from a sample of N decryptions with the same key, the attacker can construct an array $Q = (q_{i,j})$ whose elements are 1 or 0 depending on whether or not the ith MMM of the jth decryption includes the conditional subtraction. The elements $q_{i,j}$ are called *extra reduction* or *er-values*. Similarly, initialisation gives a matrix $Q' = (q'_{i,j})$: if C_j is the input to the jth decryption then the er-value $q'_{i,j}$ is associated with the calculation of $C_j^{(i)} \equiv {C_j}^i R \bmod M$ for the digit i.

5 Some Limiting Distributions

The timing attack here was first reported in [16], but precise MMM output bounds now allow a much more accurate treatment of the probabilities and hence more reliable results. More exact figures should make it easier to determine the precise strength of an implementation against the attack. Important first aims are to determine the probability of extra reductions and to establish the probability distribution function for the MMM outputs P. To this end two reasonable assumptions are made about such distributions:

x) The ciphertext inputs C behave as realizations of independent random variables which are uniformly distributed over $[0, M)$.
xi) For inputs A and B to MMM during an exponentiation, the output prior to the conditional subtraction is uniformly distributed over the interval $[ABR^{-1}, M{+}ABR^{-1})$.

Assumption (x) is fulfilled if, for example, the ciphertext is randomly chosen (typical for RSA key exchange) or message blinding has been performed as proposed in [4]. Here it is also a convenient simplification. In fact, there may be some initial non-uniformity (with respect to the Euclidean metric) which might arise as a result of content or formatting, such as with constant prefix padding. We return to this topic in Remark 1.

In (xi), the multiples of M subtracted from the partial product P during the execution of MMM are certainly influenced by *all* bits of A, B and M. However, the probability for the final conditional subtraction is essentially determined by the *topmost* bits of the arguments.

Before the formal treatment we illustrate the limit behaviour of the distributions which will be considered and provide the informal reasoning behind their construction. In order to exhibit the difference between squares and multiplications software was written to graph the probability of the extra subtraction in the limit of three cases, namely the distribution of outputs after a long sequence of i) squarings of a random input, ii) multiplications of independent random inputs, and iii) multiplications of a random input by the same fixed constant A. The result is illustrated in Figure 1. In the second case, two independent values are chosen from the kth distribution in the sequence. They are used as the inputs to MMM and the output generates the $k{+}1$st distribution. In all three cases the convergence is very rapid, with little perceptible change after 10 or so iterations.