Leakage from Montgomery Multiplication

Colin D. Walter

Comodo CA Ltd, Bradford, Yorks BD7 1HR, UK. Colin.Walter@comodo.com

1 Introduction

Modular multiplication $P = A \times B \bmod M$ is a fundamental operation in most public key cryptography. Its efficiency is usually critical in determining the overall efficiency of a system because it is the main component in modular exponentiation and in elliptic curve point multiplication. There are several algorithms which can be chosen for performing modular multiplication, of which those by Barrett [1], Montgomery [6] and Quisquater [2] are the most widely known. Most optimisations which can be applied to one modular multiplication algorithm can also be applied to the others, so that all have the same overall complexity [9]. However, Montgomery's method is rather more straight-forward to implement; generally less work is involved in achieving the optimisations.

This chapter delves into certain aspects of Montgomery's algorithm: it seeks to retain the advantages of simple and efficient code while at the same time addressing the issue of side channel leakage from the final conditional subtraction. We study the main loop and the final conditional subtraction separately in order to determine a fully precise specification for the output and hence determine how much data is leaked through the conditional subtraction side channel. This enables us to fix the leakage very satisfactorily.

2 Montgomery Reduction

Modular multiplication is really a combination of two processes: multiplication and modular reduction. These are generally *interleaved* for space efficiency reasons: this keeps the intermediate values within a very small multiple of the modulus. This is also called the *integrated* approach. With the *separated* technique, the multiplication is performed completely beforehand, allowing more time-efficient methods to be employed. This is then followed by modular reduction of the product. We start by looking at this process of reduction.

Definition 1. *Suppose positive integers M and R have no common factor. Then,*

i) A' is called the Montgomery reduction *of A modulo M with respect to R if $A' \equiv AR^{-1} \bmod M$ and $AR^{-1} \leq A' < AR^{-1}+M$.*

ii) Conversely, $\overline{A}$ is called a Montgomery representation *or* Montgomery residue *of A with respect to R if it satisfies $\overline{A} \equiv AR \bmod M$. When R is clear, $\overline{A}$ is called an M-*residue.

Note that this definition contains two different inverses of R: one is the residue modulo M which is the modular inverse of R in $\mathbb{Z}/M\mathbb{Z}$, the other is the rational number which is the fractional inverse of R in $\mathbb{Q}$. The context, indicated by the presence or absence of "mod$M$", makes the intended choice clear.

Co-primality of M and R guarantees that there is a one-to-one correspondence between a complete set of residues mod M and the set of residues $\{R'.R \bmod M \mid 0 \leq R' < M\}$. So there is a value R' such that $R'.R \equiv 1 \bmod M$, i.e. R has an inverse mod M. So the Montgomery reduction exists when M and R have no common factor, and the bounds ensure that it is unique. For cryptographic applications the co-primeness property usually holds: R is generally a power of 2 so that division by R is easily performed by shifting, whereas M is a product of some large primes, and hence odd.

The Montgomery reduction A' of A mod M can be obtained by finding integers A' and Q satisfying

$$A'R - QM = A$$

and such that Q is in the interval $[0..R[$. When R has the form $R = r^n$, the solution can be generated using the same process as in Hensel's Lemma, which solves the equation iteratively modulo higher and higher powers of r.

When A and M have representations over base (or radix) r with digits a_i and m_i respectively, the Henselian process for obtaining the Montgomery reduction A' with respect to $R = r^n$ is given in Fig. 1. There, ${m_0}^{-1}$ is the inverse of m_0 modulo r. Usually the digits occupy words of memory, so that $r = 2^k$ where k is the number of bits per word. Then the division by r in the last line of that figure is simply a shift by one position of an array of words.

The choice of digit q_i in the loop guarantees that the division by r is exact. Thus, if the digits q_i are formed into the "quotient" $Q = \sum_{i=0}^{n-1} q_i r^i$ then the output satisfies $A' = (A+QM)r^{-n}$ where $Q < r^n = R$, as required. The bounds on A' now follow, showing it is the Montgomery reduction of input A.

The term Ar^{-n} becomes smaller as n is increased. Hence, by choosing n such that $A < Mr^n$ the output satisfies $A' < 2M$ and an extra conditional subtraction of M will yield the least non-negative residue of AR^{-1} mod M. For example, if the input A were obtained as a product of two reduced residues modulo M (i.e. least non-negative) then $A < M^2$ and we would just need n such that $M < r^n$ in order to achieve an output which is also fully reduced by the extra conditional subtraction.

Function MonRed(A, M, r, n): A'
Pre-condition: M and r are co-prime.
Post-condition: $A' \equiv Ar^{-n} \bmod M$ with $Ar^{-n} \leq A' < Ar^{-n}+M$.

```
A' ← A
For i ← 0 to n-1 do
    q_i ← -a'_0 m_0^{-1} mod r
    A' ← (A'+q_i M) div r
Return A'
```

Fig. 1. Montgomery Modular Reduction.

Unlike classical modular reduction, the choice of quotient digit q_i does not depend on the most significant digit of A' but on its least significant. This means that q_i can be determined precisely without waiting for carry propagation to be completed in the previous loop iteration. This is advantageous for application in a systolic array where the processing elements perform digit level computations [10].

3 Montgomery Modular Multiplication

Instead of pre-computing the product $A \times B$, the Montgomery reduction of $A \times B$ modulo M can be obtained by interleaving the multiplication and the reduction, as in Fig. 2. We need one of the inputs, say B, to have a representation to base r with at most n digits.

Function MonPro(A, B, M, r, n): C
Pre-condition: M and r are co-prime, $B = \sum_{i=0}^{n-1} b_i r^i < r^n$.
Post-condition: $C \equiv ABr^{-n} \bmod M$ and $ABr^{-n} \leq C < M+ABr^{-n}$.

```
C ← 0
For i ← 0 to n-1 do
    q_i ← -(c_0+a_0 b_i) m_0^{-1} mod r
    C ←  (C+b_i A+q_i M) div r
Return C
```

Fig. 2. Montgomery Modular Multiplication without Conditional Subtraction.

It is easy to verify the code of Fig. 2 from its similarity to MonRed and by observing that the non-modular operations compute $A \times B$ with a shift equivalent to a factor of r^{-n}. In fact, taking $B = 1$ yields the MonRed algorithm. As in the MonRed algorithm, there is no upper bound on the value of input A. Moreover, the bound on B is in terms of n and not M. Thus there is no need to ensure the inputs are least non-negative residues modulo M.

Since the output of the ith iteration is exactly analogous to the output of the nth, for which $B < r^n$, we know that the partial product C is less

than $M+A$ throughout the calculation. This gives a bound on how large the register for C needs to be. For most applications $M < r^n$ and $A < r^n$ so that C requires one more bit than r^n. A further extra word may also be needed for intermediate results before the shift down corresponding to the division by r. A detailed time and space efficiency analysis of different methods to compute the update to the partial product C is given by Koç, Acar and Kaliski [3].

Setting $Q = \sum_{i=0}^{n-1} q_i r^i$ gives $C = (AB+QM)r^{-n}$ and so $Q \approx ABr^{-n}/M$ when C is bounded by a small multiple of M. To be precise, if $C-\delta M$ is the smallest non-negative residue, then $Q+\delta = \lfloor ABr^{-n}/M \rfloor$ yields the integer quotient. So further conditional subtractions of M from C to obtain the least non-negative residue can be combined with incrementing Q to yield the integer quotient.

The normal presentation of the algorithm includes a final conditional subtraction of M to yield an output less than M. It is omitted in this first version of Montgomery multiplication for three reasons: it is unnecessary when MonPro is used for exponentiation, it is a strong source of side channel leakage and, for non-fully reduced inputs, more than one subtraction of M may be necessary. However, there are two useful versions of Montgomery multiplication which include a final conditional subtraction. They are given in Figs. 3 and 4.

Function MonPro$^{(M)}(A, B, M, r, n)$: C
Pre-condition: M and r co-prime, $A < M$, $B < r^n$.
Post-condition: $C \equiv ABr^{-n} \bmod M$ and $C < M$.

```
C ← MonPro(A,B,M,r,n)
if C ≥ M then C ← C-M
```

Fig. 3. Montgomery Modular Multiplication with Bound M.

Function MonPro$^{(R)}(A, B, M, r, n)$: C
Pre-condition: M and r co-prime, $A<R$, $B<R$, $M<R$ for $R = r^n$.
Post-condition: $C \equiv ABr^{-n} \bmod M$ and $C < R$.

```
C ← MonPro(A,B,M,r,n)
if C ≥ R then C ← C-M
```

Fig. 4. Montgomery Modular Multiplication with Bound R.

It is easy to check that the input bounds imply the output bounds in both cases. Moreover, the bounds are such that outputs can be used as inputs to another execution of the same algorithm. This is very convenient for applications involving exponentiation. The second version, with bound R, is marginally more efficient than the first for two reasons. First, this is because the comparison is easier to implement: typically it just requires looking at

an overflow bit rather than performing a potentially full length subtraction. Secondly, the frequency of the subtractions is usually lower for the second version.

4 Exponentiation

All exponentiation algorithms consist of a sequence of products. Therefore, in order to use MonPro for modular exponentiation, it suffices to adjust for the extra power of R and to check that the output from one use of MonPro satisfies the bounds on the input required for its use in any subsequent MonPro. If

$$Z = X \times Y \bmod M$$

is one of the modular multiplications during the exponentiation when normal modular products are computed, then the usual choice for the corresponding multiplication using MonPro is

$$\overline{Z} = \text{MonPro}(\overline{X}, \overline{Y}, M, r, n)$$

which operates on the corresponding M-residues. Comparing powers of R, we find $\text{MonPro}(\overline{X}, \overline{Y}, M, r, n) \equiv \overline{X} \times \overline{Y} \times R^{-1} \equiv XR \times YR \times R^{-1} \equiv X \times Y \times R \equiv Z \times R \equiv \overline{Z} \bmod M$. Thus, the entire exponentiation is done correctly with MonPro on the corresponding M-residues if the input is adjusted to an M-residue and the output is re-adjusted back from an M-residue. An example of this is given in Fig. 5 for the square-and-multiply method of exponentiation[1].

Function $\text{MonExp}^{(R)}(T, N, M, r, n, R^{(2)}) : S$
Pre-conditions: M and r co-prime, $T<M<R$ for $R = r^n$, $R^{(2)} < R$, $R^{(2)} \equiv R^2 \bmod M$, and $N=(n_{k-1}...n_2 n_1 n_0)_2$ with $n_{k-1}=1$.
Post-condition: $S = T^N \bmod M$ and $0 \leq S < M$.

```
T̄ = MonPro^(R)(T,R^(2),M,r,n)
S̄ ← T̄
For i ← k-2 downto 0 do
      S̄ ← MonPro^(R)(S̄,S̄,M,r,n)
      If n_i=1 then S̄ ← MonPro^(R)( S̄,T̄,M,r,n)
S ← MonPro^(R)(S̄,1,M,r,n)
```

Fig. 5. Square-and-Multiply Exponentiation with $\text{MonPro}^{(R)}$.

To ensure the correct power of R in all operands, the evaluation of $S = T^N \bmod M$ requires a pre-processing step to convert T to its M-residue:

[1] Strictly speaking, M must be square-free to avoid the possibility of output $S = M$, which is forbidden in the post-condition of the code. The output bound is treated later in this section.

$$\overline{T} = \text{MonPro}(T, R^2 \bmod M, M, r, n)$$

and the post-processing step to convert back from an M-residue:

$$S = \text{MonPro}(\overline{S}, 1, M, r, n).$$

It is easy to see that the first of these introduces a factor of R, while the second removes such a factor. Corresponding similar pre- and post- processing steps also correctly introduce and eliminate a factor of R when $\text{MonPro}^{(M)}$ or $\text{MonPro}^{(R)}$ is used throughout the exponentiation instead of MonPro – simply replace MonPro by $\text{MonPro}^{(M)}$ or $\text{MonPro}^{(R)}$ as appropriate.

For the large exponents N typical of public key cryptography, these two extra modular multiplications have a negligible cost when compared with the advantages of a simpler multiplication algorithm. However, $R^{(2)} \equiv R^2 \bmod M$ also needs to be pre-computed and stored. Since M is usually fixed for many exponentiations, the time penalty for this can normally be amortized over the lifetime of the modulus.

The other main requirement for using Montgomery modular multiplication in exponentiation is that the output from one multiplication satisfies the pre-conditions for inputs to the subsequent multiplications. This is plainly the case if $\text{MonPro}^{(R)}$ is used throughout: inputs and outputs are both bounded by R. To use $\text{MonPro}^{(M)}$ throughout, the condition $M < R$ must be added in order to guarantee that the "B" input is small enough. Then all inputs and outputs are bounded by M.

Now suppose MonPro is used in the exponentiation algorithm. In order to achieve a common bound, say B, on all inputs and outputs of MonPro, it is necessary that $M+B^2R^{-1} \leq B$. This just requires that the quadratic $B^2R^{-1}-B+M$ has real roots, i.e. $4M < R$. This is achieved by choosing a large enough value for R, that is, a sufficiently large n where $R = r^n$. Then any bound B can be chosen as long as it is between the two roots. This is readily seen to be the case for bounds such as $B = 2M$ or $B = \frac{1}{2}R$, or even a weighted average $B = 2\lambda M+\frac{1-\lambda}{2}R$ where $0 \leq \lambda \leq 1$ [13].

For all three modular multiplication algorithms, the above bounds are sensible conditions on cryptographic inputs T to an exponentiation. With bound $B = M$ or $B = R$ for $\text{MonPro}^{(B)}$, the initial conditions $T < B$ and $R^{(2)} < R$ ensure that $\overline{T} < B$, making $\overline{T}$ suitable for subsequent use in the exponentiation. Thereafter, every input to $\text{MonPro}^{(B)}$ is bounded above by B, so that its output is also bound by B. For MonPro, the initial conditions $T < M$ and $R^{(2)} < R$ ensure that $\overline{T} < B$ for the acceptable bound $B = 2M$. Thus, the pre-processing input $R^{(2)}$ need not be fully reduced in any of three cases; R is an adequate bound for it in each case.

For $\text{MonPro}^{(B)}$ with bound $B = M$ or R, and for MonPro with bound $B = 2M$ or $\frac{1}{2}R$, the loop of the post-processing modular multiplication by 1 generates output S satisfying $S < M+BR^{-1} < M+1$. Hence, for none of the three algorithms does a final subtraction take place, nor is it necessary to obtain a fully reduced output, except possibly when $S = M$ occurs.

However, a loop output of $S = M$ is almost impossible. For MonPro$^{(M)}$, this is irreconcilable with the obvious properties $0 < \overline{S} < M$ and $\overline{S} \equiv 0 \bmod M$. When M is square-free, $S \equiv 0 \bmod M$ implies $T \equiv 0 \bmod M$. In this case, a pre-condition of $0 \leq T < M$ forces $T = 0$. Then the output of every modular multiplication is 0, ensuring that $S = 0$, so that $S = M$ does not occur. So exponentiation with MonPro never uses even a single subtraction to achieve a fully reduced output. Otherwise, with MonPro$^{(R)}$ or MonPro in the non-square-free case, a single subtraction may be required to obtain a fully reduced output [11][2].

5 Space and Time Comparisons

In this section, the time and space requirements of exponentiation methods using the three modular multiplication algorithms are compared: the standard MonPro$^{(B)}$ with conditional subtraction and bound $B = M$ or R, and MonPro with no subtraction and bound $B = 2M$ or $\frac{1}{2}R$.

The radix of the number representations here is $r = 2^k$ where k is the native bit length of words in the processor which performs the modular multiplication. Typical word lengths are small powers of 2, such as 8-, 16-, 32- and 64-bit. Standard key lengths for RSA normally coincide with multiples of these, such as 1024, 1536 and 2048. The same is true for many of the standard prime fields $\mathbb{F}_p$ used in elliptic curve cryptography [8]. Consequently, to achieve the property $M < R$ required for MonPro$^{(B)}$ with minimal cost, the property $R < 2M$ frequently holds as well, that is, R is the smallest power of 2 greater than M. So, discarding the final subtraction and using MonPro for exponentiation instead of MonPro$^{(B)}$ comes at an initial cost of increasing the number of iterations n, probably by just 1, to ensure $4M < R = r^n$.

So, with standard key and word lengths and the minimal choice for n, the register containing C in MonPro$^{(B)}$ needs to have one more bit or one more word than M because loop output values can be up to $M+B$ in magnitude where $B = M$ or R. Increasing n by 1 to make $4M < R$ with an input bound of $B = 2M$ or $\frac{1}{2}R$ generates intermediate values also less than $M+B$, but this is still below R. So MonPro exponentiation needs *no* extra words for the intermediate calculations. Indeed, it requires only *two* more bits for C than for M. So the loops in the modular multiplications of MonPro and MonPro$^{(B)}$ exponentiations have the same computational space requirements in a fully word-based implementation, and MonPro uses only one more register bit when the top word is reduced to contain only the bit positions that are needed.

Because both manipulate the same number of words, there is also unlikely to be any time difference between single loop iterations in MonPro and MonPro$^{(B)}$. The topmost incomplete word or equivalent individual bits cannot be processed any faster than full words because the clock speed is set to that of the slowest word operation.

[2] *see* Exercise 2 for the non-square-free case.

So the main time and space differences will be between the final conditional subtraction when MonPro$^{(B)}$ is used for exponentiation and the extra loop iteration when MonPro is used. A leak-resistant implementation of MonPro$^{(B)}$ exponentiation will always perform the subtraction, but needs an extra register to hold the pre-subtraction value of C (the "minuend") as well as its post-subtraction value. On the other hand, being more complex than a subtraction, the extra iteration of MonPro may require more time. Taking into account their relative complexity, the loop iteration is most likely to take the same time as a subtraction or double that time since the clock frequency will probably be chosen to make word-level multiply-accumulate and subtraction operations take the same time. At a theoretical level, the subtraction itself might be equivalent to about half an extra loop iteration, and selecting the result of the subtraction or its minuend equivalent to another half of a loop iteration. This would make MonPro and MonPro$^{(B)}$ take essentially the same time.

The subtractor itself may require significant extra dedicated hardware and associated data manipulation require extra time. Moreover, code size will be increased by having to incorporate instructions for the subtraction. As observed in §4, MonPro exponentiation requires no final subtraction, so that the extra hardware and code may not be necessary, although it is likely that other cryptographic operations on the chip will require them.

Although the precise cost will be implementation specific, this discussion indicates that using MonPro$^{(B)}$ with its conditional subtraction will be more expensive in hardware than using MonPro for exponentiation, and the time requirements are essentially identical. In conclusion, exponentiation using MonPro with $4M < R$ and no final subtraction is a cost effective and straight-forward solution to the problem of side channel leakage from conditional subtractions.

6 Side Channel Analysis

A substantial embarrassment to many smart card manufacturers in the '90s was the public discovery that naïve implementations of Montgomery's algorithm can cause substantial side channel leakage, enabling private keys to be recovered from fewer than a hundred uses of the key [4, 12]. The main problem arises from the conditional subtraction in MonPro$^{(M)}$ and MonPro$^{(R)}$ which, because of the length of keys, takes a large number of clock cycles to complete. It is therefore very evident in any EMR or power trace.

Nowadays there are many effective counter-measures to prevent such leakage, not least of which is using MonPro for exponentiation rather than MonPro$^{(M)}$ or MonPro$^{(R)}$. Another counter-measure is to modify MonPro$^{(M)}$ and MonPro$^{(R)}$ slightly so that the subtraction is always performed. Then the original loop output value C is kept, and the new or old value is selected

according to the sign of the new value. In this way the leakage is considerably reduced. Other counter-measures are discussed in other chapters; here we limit ourselves to measuring the leakage from $\text{MonPro}^{(M)}$ and $\text{MonPro}^{(R)}$ and varying the parameter R in order to minimise it.

Assume that an implementation of exponentiation using a public modulus and private exponent is under attack, and that all conditional subtraction events can be observed clearly through a side channel. First, we make the simple observation that when the conditional subtraction occurs in one modular multiplication but not in another, then the multiplications must involve different arguments. Suppose an attacker can choose the input to an exponentiation with the secret key on the target device and he can observe individual conditional subtractions. With knowledge of the public modulus and exponentiation algorithm, he can also write a software simulation of the exponentiation which will generate the same sequence of conditional subtractions when it has the same input and uses a correct guess at the secret key. He then guesses the bits of the key in the order that they are consumed by the algorithm. Whenever there is a difference between the conditional subtractions in the side channel leakage and his simulation, he knows the operands at that point differ between the two exponentiations. So he has guessed incorrectly and he backtracks to change his most recent guesses; several previous bits may need to be adjusted. Providing the subtractions occur with probability not too close to 0 or 1, he has a good chance of recovering the whole of the private key in this way[3]. Such an attack uses leakage from a single exponentiation and can be applied to both RSA and ECC, as well as other exponentiation-based protocols. The obvious counter-measure is to blind the input text T before exponentiating.

Secondly, in elliptic curve cryptography, the classical formulae for point addition and point doubling are so different that it is easy to distinguish them in side channel traces. Then, with an algorithm such as square-and-multiply, it becomes trivial to read the pattern of adds and doubles and deduce the secret key. One counter-measure is to use "unified" formulae for both doubling and adding, so that the same sequence of field operations is performed in both cases. Unfortunately, for a point doubling some pairs of these operations have identical arguments, whereas they are different for point additions. When $\text{MonPro}^{(M)}$ or $\text{MonPro}^{(R)}$ is used to implement the associated modular arithmetic, a difference in the behaviour of the conditional subtractions indicates a point addition unequivocally, whereas identical behaviour makes a point doubling more likely to be the case. With the shorter keys which occur in ECC, the attacker is left with a small search space of possible keys which it is often computationally feasible to traverse [15]. This attack does not require the opponent to be able to input data of his own choosing to the exponentiation although it may require repeated use of the key until a sufficiently favorable exponentiation occurs.

[3] In the next section we find that the probability of a subtraction is at most $\frac{1}{2}$ and, by increasing R, the probability can be made as close to 0 as desired.

Thirdly, a number of side channel attacks depend on recording the frequency of the conditional subtraction at different points in an exponentiation algorithm over a number of executions of it with different data and the same unblinded (secret) exponent. In particular, as we see in the next section, the frequency is different for squarings and multiplications. Hence repeated use of the square-and-multiply algorithm with MonPro$^{(R)}$ (Fig. 5) or MonPro$^{(M)}$, the same secret exponent and random input data T, would enable the sequence of squarings and multiplications to be deduced, and hence the bits of the secret key obtained.

From this summary of attacks, it is clear that no implementation of Montgomery modular multiplication or reduction should be allowed a final conditional subtraction which takes execution time which is not constant. The only exception is if the subtraction is extremely rare – a case that may actually arise below for certain choices of the parameters. Otherwise MonPro$^{(M)}$ and MonPro$^{(R)}$ should always perform their subtraction and choose the original or updated value as appropriate.

7 Frequencies of Conditional Subtractions

In this section we consider a set $\mathcal{S}$ of executions of MonPro$^{(M)}$ with common modulus M. The aim is to estimate the expected frequency of the conditional subtraction for the main ways in which the set might have arisen. Then side channel leakage about the conditional subtractions can be used to deduce which of the causes is the most likely for $\mathcal{S}$. The set might represent corresponding operations in a number of different exponentiations using the same unblinded key, and the aim may be to determine if these operations were all multiplications or all squarings. Similar results can be obtained for MonPro$^{(R)}$, but they are considerably complicated by having non-uniform distributions for the inputs and outputs: inputs and outputs have ranges greater than M, so that some residues classes modulo M have more than one representative.

As an example, consider the set of all MonPro$^{(M)}$ multiplications $A\times B$ mod M for M=7 and R=8. Since $R^{-1} \equiv 1 \bmod M$, the output C satisfies $C \equiv AB \bmod M$. So, with ABR^{-1} as a lower bound, the outputs from the MonPro loop are those given in Table 1. The overall frequency of the conditional subtraction is $\frac{5}{49}$. However, restricting to squares $A\times A$ mod M, the frequency of the subtraction becomes $\frac{1}{7}$, which is a little greater. If the input A is fixed to A=5, then the probability of the subtraction for random B rises to $\frac{2}{7}$. Different fixed values of A result in other quite different probabilities.

There are three main types of set $\mathcal{S}$ to consider: those arising from squaring, those arising from multiplications in which both arguments are free to assume any values independently, and those arising from multiplications in which one argument has the same value for the whole set. There are other sets of interest which are not discussed. For example, none of the above matches

0	0	0	0	0	0	0
0	1	2	3	4	5	6
0	2	4	6	1	3	5
0	3	6	2	5	**8**	4
0	4	1	5	2	6	3
0	5	3	**8**	6	4	**9**
0	6	5	4	3	**9**	**8**

Table 1. A Modular Multiplication Table: unreduced outputs from MonPro(A, B) when $M = 7, R = 8$. The five bold entries require reduction.

the set of all multiplications from a single m-ary or sliding windows left-to-right exponentiation (unless $m = 2$): because one input, say A, is taken from a set of only $\log_2 m$ distinct multipliers, it is neither constant nor uniformly distributed.

Several reasonable, simplifying assumptions are made in order to derive the frequencies of the subtraction for $\mathcal{S}$. They are often very hard to justify theoretically, but several are closely related to the diffusion properties on which the associated cryptography relies. First,

- it is assumed that $\phi(M) \approx M$.

The "Euler phi" function having a value close to M means that M is a product of a small number of large, not necessarily distinct primes, as in the case of RSA and $\mathbb{F}_p$. The property $\phi(M) \approx M$ just states that almost every number is prime to M.

Suppose input A of MonPro is prime to M. Then A has an inverse modulo M. Therefore, if the other argument B has a uniform distribution modulo M, so will the output C. So at least one input being uniformly distributed means that, to a very good approximation, the output is also uniformly distributed. Such uniformity is propagated from input to output through every instance of MonPro in an exponentiation if the initial input is uniformly distributed. Due to formatting and construction restrictions, the inputs to the exponentiation may not be uniform in practice, but diffusion occurs so rapidly during exponentiation that, except possibly for the initial one or two multiplicative operations, the uniformity can be assumed.

Let A, B and Z be discrete random variables over the interval of integers $0..M-1$ corresponding respectively to the two MonPro$^{(M)}$ inputs and the variation in output C of the MonPro loop within the interval $[ABr^{-n}, M+ABr^{-n}[$. Suppose

- A, B and Z are all independent and uniformly distributed

and let π_{mu} be the probability that the final subtraction takes place. Then $\pi_{mu} = pr(Z+ABR^{-1} \geq M) = \frac{1}{M^3} \sum_{Z=0}^{M-1} \sum_{A=0}^{M-1} \sum_{B=0}^{M-1} (Z+ABR^{-1} \geq M) \approx \frac{1}{M^3} \sum_{A=0}^{M-1} \sum_{B=0}^{M-1} ABR^{-1} \approx \frac{1}{M^3} \int_0^M \int_0^M ABR^{-1} dA\, dB$ where $Z+ABR^{-1} \geq M$ is 0 or 1 according to the truth of the inequality. (The approximations

arise from using real numbers instead of integers, and are very accurate for cryptographic sized moduli.) So

$$\pi_{mu} \approx \frac{1}{4} MR^{-1} \tag{1}$$

This is the probability of the subtraction for a set $\mathcal{S}$ of multiplications under the above hypotheses. Suppose the ith operation in a set of exponentiations is always a multiplication and that the inputs to the exponentiations are uniformly distributed modulo M. Then π_{mu} is the probability of a subtraction when $\mathcal{S}$ is the corresponding set of instances of MonPro$^{(M)}$.

Now let π_{sq} be the probability that the final subtraction takes place when MonPro$^{(M)}$ is used to square a uniformly distributed input. For the same definitions as above, suppose

- A and Z are independent and uniformly distributed.

Then $\pi_{sq} = pr(Z+A^2R^{-1} \geq M) = \frac{1}{M^2} \sum_{Z=0}^{M-1} \sum_{A=0}^{M-1} (Z+A^2R^{-1} \geq M) \approx \frac{1}{M^2} \sum_{A=0}^{M-1} A^2R^{-1} \approx \frac{1}{M^2} \int_0^M A^2R^{-1} dA$, whence

$$\pi_{sq} \approx \frac{1}{3} MR^{-1} \tag{2}$$

Unlike sets of multiplications, in practice most sets of squarings satisfy the criteria to apply this value for π_{sq}. Since $\pi_{mu} < \pi_{sq}$, the subtraction is less frequent for multiplications than for squarings, as in the example with M=7.

Now suppose the value of A is fixed but the argument B is uniformly distributed on $0..M-1$. Let π_A be the probability that the conditional subtraction takes place. For definitions as before, assume also that

- B and Z are independent and uniformly distributed.

Then $\pi_A = pr(Z+ABR^{-1} \geq M) = \frac{1}{M^2} \sum_{Z=0}^{M-1} \sum_{B=0}^{M-1} (Z+ABR^{-1} \geq M) \approx \frac{1}{M^2} \sum_{B=0}^{M-1} ABR^{-1} \approx \frac{1}{M^2} \int_0^M ABR^{-1} dB$. Hence

$$\pi_A = \frac{1}{2} AR^{-1} \tag{3}$$

So, for fixed multiplier A, the frequency of subtractions depends strongly on its size. For large A, such as 5 and 6 in the example with $M = 7$, the frequency is highest. At the other extreme, note that the value is correct even for $A = 0$, although it is not prime to M and so causes the output not to be uniformly distributed. As expected, the average value of π_A is π_{mu} when A is uniformly distributed.

8 Variance in Frequencies & SCA Errors

If the frequency of the conditional subtraction is used to determine whether the set $\mathcal{S}$ consists of multiplications or squarings, then the accuracy of the decision is important to know.

Let $\mathcal{S}_i$ be the set of ith modular multiplications in a collection of t square-and-multiply exponentiations using the same 1024-bit key. So $|\mathcal{S}_i| = t$. There are about 1500 sets $\mathcal{S}_i$ to classify in order to recover the bit pattern of the secret exponent. If e errors are made, then, with a simple-minded approach in which every bit might be one that has been mis-classified, approximately 1500^e different alternatives might have to be tried before the correct exponent is discovered. Of course, operations for sets $\mathcal{S}_i$ with frequencies close to the average of π_{mu} and π_{sq} are the most likely to be mis-classified, and the search should begin there. This would find the correct key much more quickly. Nevertheless, it is clear that the error count e has to be kept very small for this to become the foundation of a computationally feasible attack.

Assuming the inputs to the t exponentiations are independent, the conditional subtractions should occur independently within each $\mathcal{S}_i$. Then the behaviour is that of a binomial random variable for t trials with probability $p = \pi_{mu}$ or π_{sq}. The expected number of subtractions is tp and its variance is $\sigma^2 = tp(p-1)$. To make use of tables for the normal distribution, it is more likely that we will prefer to work with the probability of the subtraction rather than its total count. In this case, the mean is p and the variance is

$$\sigma_{mu}^2 = \frac{1}{4t} MR^{-1}(1-\frac{1}{4}MR^{-1})$$

$$\sigma_{sq}^2 = \frac{1}{3t} MR^{-1}(1-\frac{1}{3}MR^{-1})$$

or

$$\sigma_{A}^2 = \frac{1}{2t} AR^{-1}(1-\frac{1}{2}AR^{-1})$$

The point at which classification as a squaring or multiplication is equally likely is the weighted average

$$\pi = \frac{\pi_{sq}\sigma_{mu} + \pi_{mu}\sigma_{sq}}{\sigma_{mu} + \sigma_{sq}} \tag{4}$$

If the conditional subtraction occurs less often than this in $\mathcal{S}_i$ then the set probably contains multiplications; if it is greater then the set probably contains squarings. The probability of making an incorrect decision can be obtained by looking up tables for the normal distribution, which approximates the binomial distribution when t is large. It uses the fact that if X is a normal random variable with mean μ and variance σ^2, then $\sigma^{-1}(X-\mu)$ has the N(0,1) distribution, which is tabulated [7]. So, the probability of the count being on the wrong side of π is easily seen to be approximately $Pr(Z > \frac{\pi_{sq}-\pi_{mu}}{\sigma_{mu}+\sigma_{sq}})$, where Z is an N(0,1) random variable. This is the average probability of misclassifying an operation, but clearly the operation is much more likely to be correctly classified when its subtraction probability is well away from π than when it is close to π.

This error probability is roughly $Pr(Z > \frac{\sqrt{t}}{13.2})$ for a typical value of $MR^{-1} \approx \frac{3}{4}$. Then $t > 1800$ would lead to less than 1 error in the 1500

or so operations for a 1024-bit exponentiation. If R is doubled, the value of interest in the N(0,1) tables is divided by about $\sqrt{2}$. Then around 18 errors appear. More refined attacks on the same data can deduce the key with many fewer exponentiations [14]. However, as the example shows, a very suitable counter-measure is to reduce MR^{-1} by increasing R, because this reduces the incidence of conditional subtractions, and hence reduces the side channel leakage.

9 A Surprising Improvement

The formulae for π_{mu} and π_{sq} show that one of the easiest way to reduce leakage is to reduce MR^{-1}. Since the bit length of M is generally fixed, this means increasing the number of iterations n in the Montgomery Modular Multiplication algorithm MonPro (Fig. 2). This reduces the frequency of conditional subtractions. However, as noted in the last paragraph of §4, MonPro can be used for exponentiation on its own without the final conditional subtraction once $MR^{-1} < \frac{1}{4}$. This is because the extra shifting down then makes the output small enough for re-use as an input to the next occurrence of MonPro.

A straightforward solution to side channel leakage would be to reduce standard key lengths by 2 bits. Then $4M < R$ would hold automatically for the minimum choice of n in a word-based implementation. Consequently, MonPro could be used throughout an exponentiation without the need for extra iterations or conditional subtractions, and all intermediate values would remain within the word boundaries. However, suppose that M reaches the top of the word boundary, so that MonPro requires one more iteration than MonPro$^{(M)}$ or MonPro$^{(R)}$. We will now look in more detail at how this affects the bounds on intermediate values when MonPro is used.

So, assume also that the word size is at least 2 bits and that n is chosen minimally to ensure $4M < R$. This means $r \geq 4$ and $rM < R < 2rM$. Let R' denote the Montgomery multiplier which the standard version MonPro$^{(M)}$ or MonPro$^{(R)}$ would have used. So $R = rR'$ and $M < R' < 2M$. This standard version produces loop outputs bounded by $ABR'^{-1}+M$, which is only just larger than R'. But here MonPro performs an extra division by r. Consequently, the upper bound $ABR^{-1}+M$ on the output is less, making it more likely to satisfy $ABR^{-1}+M < R'$. An interesting question is therefore whether all inputs and outputs would be bounded by R' when the extra iteration is performed, because the hardware requirements would then be lower. Substituting in the bound R', the desired property holds if $R'r^{-1}+M < R'$, i.e. if $M < R'(1-r^{-1})$. Writing this as $M < R'r^{-1}(r-1)$ shows the condition is just that the top word of M is not $r-1$, i.e. not entirely 1s.

Theorem 1. ([13], Thm 5.) *Suppose the top digit of M is not $r-1$, that $r \geq 4$, and that MonPro is executed with n iterations where $M < R' = r^{n-1}$, i.e. one more iteration than normal. Then inputs bounded by R' generate outputs bounded by R'.*

In other words, in almost all cases M is such that a single extra iteration of the MonPro loop avoids any need for a conditional subtraction. In effect, it is equivalent to using MonPro$^{(R')}$ but without the conditional subtraction side channel. Intermediate values of C at the end of a loop iteration in MonPro are bounded above by $A+R'$ so that, as usual, one extra register bit is needed.

If one is prepared to exclude the rare cases of inconvenient moduli M with top digit $r-1$, this means that exponentiation can be carried out successfully using MonPro with the usual register sizes and omitting conditional subtractions, providing only that an extra iteration is done in the MonPro loop. Of course, at the end of the exponentiation there is also no need for a conditional subtraction: as noted before, the adjusting MonPro multiplication by 1 reduces the output to less than M.

With the extra iteration, the exceptional cases can be included simply by restoring the conditional subtraction, i.e. by using MonPro$^{(M)}$ or MonPro$^{(R')}$ with an incremented value of n. Then, as in §7, equation (3), the output can be assumed to be uniformly distributed modulo M, so the probability of exceeding M or R' is at most $\frac{1}{2}r^{-1}$. With a typical key lifetime of 10000 uses, and the difficulty of capturing that number of side channel traces, we just require 16-bit or larger words for the conditional subtraction to have negligible probability of happening even once per key bit during the lifetime of the key. This effectively eliminates the side channel leakage since there is insufficient data at the information theoretic level to reconstruct the key. To summarise, MonPro$^{(B)}$ (B=M or R') can be used securely for exponentiation with all keys if an extra iteration is performed and a 16-bit or larger processor is used.

10 Conclusion

This chapter has carefully derived input and output bounds for several versions of Montgomery modular multiplication in order for it to be of use in exponentiation. Some pre- and post- processing is necessary, and the precomputation of a multiplier $R^{(2)}$ whose one-time cost can be amortised over the lifetime of the modulus M.

This analysis enabled the frequencies of conditional subtractions to be deduced accurately, showing that there could be considerable side channel leakage from the ability to observe the subtractions.

It was shown how this side channel could be closed by increasing the number of iterations in the MonPro algorithm. At the further cost of one extra register bit, or the exclusion of some extreme modulus values, the conditional subtraction can be totally eliminated. However, even if the conditional subtraction were retained to include the exceptional cases, the extra iteration reduces the side channel to an unusable level except for these extreme moduli on hardware with a very small word size.

The chapter therefore provides several choices for implementing Montgomery modular multiplication in a manner which effectively eliminates any side channel leakage emanating from the final, conditional subtraction.

11 Exercises

1 i) Find 16^{-1} mod 11. Using the output bounds given in Fig. 2, compute a complete table of output from MonPro when $M = 11$, $R = 16$, and A and B are least non-negative residues modulo M.

ii) Use this table to determine the probability of the conditional subtraction in MonPro$^{(M)}$ for *a*) multiplications and *b*) squarings which have uniformly distributed inputs. *c*) For each value of A calculate the probability of the subtraction for multiplications by fixed A. *d*) Compare these probabilities with the theoretical ones derived in §7.

iii) Repeat (ii) with MonPro$^{(R)}$ in place of MonPro$^{(M)}$ (including part (*d*)).

iv) Compute a complete table of output from MonPro when $M = 7$, $R = 32$, and A and B are strictly bounded above by $2M$. What is the largest value in the table? For each value $B = M, M+1, ..., 2M-1$, if both inputs are strictly less than B, find out how many outputs are B or larger.

2 i) Let $M = 9$, $T = 3$, $N = 2$, $R = 16$ in the computation of $S = T^N$ mod M using MonPro$^{(R)}$. Find R^{-1} mod M and determine the two possible values for $R^{(2)}$. Deduce the values of $\overline{T}$ and $\overline{S}$. Hence show that the output S still requires a final subtraction to be fully reduced. Why does this not contradict the claim in the text that final subtractions are unnecessary?

ii) Repeat (i) for the same inputs but with MonPro and $R = 64$.

iii) Prove that using MonPro or MonPro$^{(R)}$ to compute $S = 3^N$ mod 9 will always give output $S = 9$ when $N \geq 2$ and R satisfies the usual constraints.

iv) Suppose P is a prime such that P^2 divides M, $T = M/P$ and $N \geq 2$. With the usual requirements on R, prove that using MonPro or MonPro$^{(R)}$ to compute $S = T^N$ mod M will always give output $S = M$.

3) Let $N =$ DDMMYY be today's date (a pseudo-random exponent), M the first prime greater than the number of this page, R the first number greater than $\frac{9}{8}M$ which is prime to M, and chosen input text $T = \lfloor \frac{1}{4}M \rfloor$.

i) Use a calculator to compute a value for $R^{(2)}$ and hence calculate the preprocessing value $\overline{T}$ given by MonPro$^{(M)}$ for an exponentiation.

ii) Convert N into binary and hence list the operations required to perform a left-to-right square-and-multiply exponentiation with exponent N. Use this list to create a list of the associated values of $\overline{S}$ when MonPro$^{(M)}$ is used to evaluate T^N mod M. Include in the list information about whether or not the

conditional subtraction occurs.

iii) Now, using the values for $\overline{T}$, R and M, reconstruct all the possible exponents which generate the same list of the conditional subtractions as N. You should create a binary tree of options for the bits, and traverse the tree systematically, calcuating the corresponding values of $\overline{S}$, and checking to see whether or not the behaviour of the conditional subtraction is the same. When the behaviour of the subtraction is different, the branch can be pruned as it cannot represent the true value of N.

iv) Count the number of nodes in the tree which have been visited. Build an argument to justify that this number is linear in the length of the exponent. Is such a search computationally feasible for a large exponent? What would happen to this count if a) R/M were larger? or b) T/M were larger?

12 Projects

1) Choose any modulus M with no factors less than 2^8, say, such that $(M-1)^2 \bmod M$ can be computed correctly and easily in the machine arithmetic that is available to you. Select a Montgomery constant $R > M$. R need not be a power of 2 for this exercise, but must be prime to M.

i) Write a program to perform exponentiation with 16-bit exponents using $\text{MonPro}^{(M)}$.

ii) Check that your code performs correctly by comparing results with a classical implementation of modular exponentiation.

iii) Check also that no final subtraction is required to get output less than M.

iv) Modify your code to collect data about the occurrences of the conditional subtraction in the ith modular multiplication for each i. (Clearly $1 \leq i < 32$.)

v) Generate $t = 10^3$ random exponentiations using the same modulus and same exponent, and collect the conditional subtraction frequencies for each position i in the exponentiation.

vi) Verify that the frequencies match those expected from §7, at least on average.

vii) Compute the means and standard deviations of the frequencies for *a*) the multiplications and *b*) the squarings. Hence compute the value of π given in equation (4).

viii) Use your value for π to partition the operations into multiplications and squarings. How many operations are mis-classified in this way? If the number of multiplications is known and used as the partitioning point, how many are mis-classified then?

ix) Repeat (v)-(viii) several times for different numbers of exponentiations to see how the number of mis-classified operations varies with t.

2 i) Repeat project 1 with MonPro$^{(R)}$ in place of MonPro$^{(M)}$, this time selecting R such that $(R-1)^2$ mod M is easily computable on your available machine.

ii) Divide the interval $[0, R]$ into, say, 50 or 100 sub-intervals, and modify the exponentiation code to collect the frequencies for the output of MonPro$^{(R)}$ lying in each sub-interval. Plot these frequencies on a graph for several different values of M such that $\frac{1}{2}R < M < R$. Explain why the curve representing these frequencies rises from 0 to $R-M$, is horizontal between $R-M$ and M, and then falls from M to R.

iii) Modify your data collection in (ii) to separate the sub-interval frequencies for multiplication outputs from those of squaring outputs. Are the graphs of these frequencies different? If so, attempt to explain the difference.

3) Collect subtraction frequencies from the exponentiation software from project 1 or 2 when the same modulus and exponent are used for a set of $t = 10^3$ exponentiations.

i) Use the length of the exponent (16 here) and the total number of modular multiplications in an exponentiation to deduce the number m of multiplications and the number s of squarings.

ii) Let $\mathcal{M}$ be the set consisting of the m operations with the lowest frequencies for the conditional subtraction, and let $\mathcal{S}$ be the complementary set consisting of the other s operations. As before, count how many operations are mis-classified if those in $\mathcal{M}$ are assumed to be multiplications, and those in $\mathcal{S}$ are assumed to be squarings.

iii) Modify your software to collect the data from the ith operation as a t-dimensional vector over {0,1} where 0 indicates that no subtraction took place, and 1 that a subtraction did take place. Call the vector v_i. For each operation, compute the average Hamming distance between v_i and the vectors in $\mathcal{M}$. Does this distance depend on whether the operation is a multiplication or a squaring?

iv) Suppose the m operations for which the distance of part (iii) is smallest represent multiplications. Re-allocate all the operations in $\mathcal{M}$ and $\mathcal{S}$ under this assumption. Does this decrease the number of errors?

v) Repeat the re-allocation process of part (iv) a number of times, and observe whether the number of errors decreases or increases with each iteration of the process.

vi) Repeat a similar re-allocation process on the initial partition of part (ii) using the average Hamming distance of v_i from the vectors in set $\mathcal{S}$.

vii) Inconsistencies between the results of (v) and (vi) must represent allocation errors in one or other case. However, are there any operations which are still incorrectly allocated? If not, reduce the value of t to see when errors start

appearing. If so, increase t to see if the number of errors decreases. Also, how does the number of inconsistencies vary with t?

viii) Repeat all the above for exponents with a variety of larger and larger lengths. Does the proportion of errors change as the length increases when t is fixed? How does the total number of errors vary for fixed t as the length varies?

4) Following on from exercise 2, investigate more thoroughly the cases where the computation of $S = T^N \bmod M$ using $\text{MonPro}^{(M)}$ might require a final conditional subtraction in the post-processing phase. Look first at when M is a prime power, then at more general non-square-free M.

References

1. P. D. Barrett, *Implementing the Rivest Shamir Adleman public key encryption algorithm on standard digital signal processor*, Advances in Cryptology – Crypto '86, Springer, 1987, 311–323.
2. J.-J. Quisquater, *Presentation at the rump session of Eurocrypt '90.*
3. Ç. K. Koç, T. Acar & B. S. Kaliski, Jr., *Analyzing and Comparing Montgomery Multiplication Algorithms*, IEEE Micro, **16**(3), 1996, 26–33.
4. P. Kocher, *Timing attack on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology – Crypto '96, N. Koblitz (editor), LNCS **1109**, Springer-Verlag, 1996, 104–113.
5. P. Kocher, J. Jaffe & B. Jun, *Differential Power Analysis*, Advances in Cryptology – Crypto '99, M. Wiener (ed.), LNCS **1666**, Springer-Verlag, 1999, 388–397.
6. P. L. Montgomery, *Modular Multiplication without Trial Division*, Mathematics of Computation, **44**(170), 1985, 519–521.
7. NIST/SEMATECH, *Cumulative Distribution Function of the Standard Normal Distribution* §1.3.6.7.1 in the "e-Handbook of Statistical Methods" at http://www.itl.nist.gov/div898/handbook/, 2006.
8. *Digital Signature Standard*, Appendix 6 (July 1999), Federal Information Processing Standard (FIPS) 186-2, NIST, Jan 2000.
9. S. E. Eldridge & C. D. Walter, *Hardware Implementation of Montgomery's Modular Multiplication Algorithm*, IEEE Trans. Comp. **42**, 1993, 693–699.
10. C. D. Walter, *Systolic Modular Multiplication*, IEEE Trans. Comp. **42**, 1993, 376–378.
11. C. D. Walter, *Montgomery Exponentiation Needs No Final Subtractions*, Electronics Letters, **35**(21), October 1999, 1831–1832.
12. C. D. Walter & S. Thompson, *Distinguishing Exponent Digits by Observing Modular Subtractions*, Topics in Cryptology – CT-RSA 2001, D. Naccache (editor), LNCS **2020**, Springer-Verlag, 2001, 192–207.
13. C. D. Walter, *Precise Bounds for Montgomery Modular Multiplication and Some Potentially Insecure RSA Moduli*, Proceedings of CT-RSA 2002, LNCS **2271**, Springer-Verlag, 2002, 30–39.
14. C. D. Walter, *Longer Keys may facilitate Side Channel Attacks*, Selected Areas in Cryptography – SAC 2003, LNCS **3006**, Springer-Verlag, 2004, 42–57.

15. C. D. Walter, *Simple Power Analysis of Unified Code for ECC Double and Add* Proceedings of CHES 2004, LNCS **3156**, Springer-Verlag, 2002, 191–204.