Some Security Aspects of the Mist Randomized Exponentiation Algorithm

Colin D. Walter[⋆]

Comodo Research Laboratory
10 Hey Street, Bradford, BD7 1DQ, UK
colin.walter@comodo.net

Abstract. The Mist exponentiation algorithm is intended for use in embedded crypto-systems to provide protection against power analysis and other side channel attacks. It generates randomly different addition chains for performing a particular exponentiation. This means that side channel attacks on RSA decryption or signing which require averaging over a number of exponentiation power traces become impossible. However, averaging over digit-by-digit multiplication traces may allow the detection of operand re-use. Although this provides a handle for an attacker by which the exponent search space might be considerably reduced, the number of possible exponents is shown to be still well outside the range of feasible computation in the foreseeable future.

Key words: Randomary exponentiation, Mist exponentiation algorithm, division chains, addition chains, power analysis, DPA, DEMA, blinding, smartcard.

1 Introduction

Because smartcards have very limited scope for the inclusion of physical security measures, the prevalence of side channel leakage from embedded cryptographic systems creates the need for new algorithms which can be implemented in more secure ways than those currently in use. This is particularly true for exponentiation, which is a major process in many crypto-systems such as RSA, Diffie-Hellman and ECC. Initial power attacks required averaging over a number of exponentiations in order to reduce the effects of noise and dependence on uninteresting data [3], [4]. Although the necessary alignment of power traces can be made more difficult by the insertion of obfuscating, random, non-data-dependent operations, the data transfers between operations usually reveal the commencement of every long integer or elliptic curve operation very clearly in each individual trace. So it is usually possible to perform the averaging process and mount an attack to extract meaningful secret data. Fortunately, attacks which require such averaging can be defeated by modifying the exponent d to $d{+}rg$ where r is a random number and g is the order of the (multiplicative) semigroup in which the exponentiation is performed [3]. This results in a different exponentiation procedure being performed every time.

[⋆] Work started while the author was at the Computation Department, UMIST, Manchester, UK

However, the author showed recently [9] that there were strong theoretical grounds for believing that, given the right monitoring equipment [6,7,1], it would be possible to break the normal m-ary exponentiation method [2] and related sliding windows techniques using a single exponentiation. This method averages over digit-by-digit products instead and relies on being able to use such averaged traces to recognise the same operands being re-used over and over. These operands are pre-computed powers of the initial text, and their use reveals the secret exponent digit. Such an attack requires no knowledge of the modulus, the input text or the output text. It would render useless the choice of $d{+}rg$ as a counter-measure, even for the case of $m = 2$, namely the standard binary "square-and-multiply" algorithm.

Random modifications have been proposed for the *algorithm* as well as the *arguments* of exponentiation in order to overcome these problems. The main suggestions are suitable where the multiplicative (additive) inverse is easily computed, such as in elliptic curve systems. Oswald and Aigner [5] have given one such example. For integer RSA, a novel exponentiation algorithm[1] called "Mist" was presented at RSA 2002 [11]. This seems to avoid all of the above-mentioned pitfalls and inverses are not required. It is more time efficient than the standard binary method when squares and multiplies have equal computational cost and it is comparable in space usage to 4-ary exponentiation. It can also be combined with any counter-measures which modify the arguments. The algorithm relies on the generation of random addition chains [2] which determine the operations to be performed, and it is based on previous work by the author [8] for finding efficient exponentiation schemes using division chains.

The Mist algorithm was created to defeat power analysis attacks which are able to detect the re-use of arguments. [11] considered only efficiency issues for the algorithm. The main aim here is to look at security issues, and, in particular, to establish that knowledge of operand re-use does not significantly reduce the effectiveness of the algorithm against power analysis or other similar attacks. Although information about operand re-use provides a handle which prunes a search tree for exponents considerably, it is still computationally infeasible to recover a secret RSA key in this way unless very significant secret data is obtained from other sources. A more likely scenario is that the attacker can only distinguish squares from multiplies. Then the search space is vastly larger, and so the algorithm appears to provide even more security.

2 The Mist Algorithm

For notation, let us assume that plaintext $P = C^D$ has to be computed from ciphertext C and secret key D. m will always represent a "divisor" in the sense of [8], and d a residue modulo m, but here these are viewed as base and digit values respectively in a representation of D. A set of allowable bases m is chosen in advance (it will be $\{2, 3, 5\}$ here), and an associated table of addition chains

[1] Comodo Research Lab has filed a patent application is respect of this [10].

for raising to the power m is stored in memory. Several variables are used: there are at least three for long integers which contain powers of C, namely Q, *TempC* and P. Of these, *TempC* is for temporary storage when Q is being raised to the power m, and so does not occur explicitly in the following code, and P contains the accumulating required output. D is updated to contain the power to which Q still has to be raised before the exponentiation is complete.

The Mist Exponentiation Algorithm [11]

```
{ Pre-condition: $D \geq 0$ }
Q ← C  ;
P ← 1  ;
While D > 0 do
Begin
    Choose a random "base" m ;
    d ← D mod m ;
    If d ≠ 0 then P ← Q^d×P ;
    Q ← Q^m ;
    D ← D div m ;
    { Loop invariant: $C^{D.Init} = Q^D \times P$ }
End ;
{ Post-condition: $P = C^{D.Init}$ }
```

Example. For D=235, m=3 yields d = (235 mod 3) = 1 and reduces D to (235 div 3) = 78. Then m=2 would give d=0 and D=39. Next, m=5 produces d=4 and D=7; m=2 gives d=1 and D=3; m=3 generates d=0 and D=1. Then, finally, m=2 yields d=1 and so D becomes 0. The pairs (m, d) are:

(3,1), (2,0), (5,4), (2,1), (3,0) and (2,1).

The corresponding powers of C contained in the variables (Q, P) are then:
(C^1, C^0); (C^3, C^1); (C^6, C^1); $(\{C^6\}^5, \{C^6\}^4 C^1) = (C^{30}, C^{25})$;
$(\{C^{30}\}^2, \{C^{30}\}^1 C^{25}) = (C^{60}, C^{55})$; $(\{C^{60}\}^3, \{C^{60}\}^0 C^{55}) = (C^{180}, C^{55})$;
$(\{C^{180}\}^2, \{C^{180}\}^1 C^{55}) = (C^{360}, C^{235})$. □

When the base set consists of the single base 2, the method simplifies to the binary square-and-multiply algorithm in which the least significant exponent bit is processed first. In general, for fixed m, the algorithm simplifies to m-ary exponentiation but performed from right to left rather than from left to right. Since the base m is varied randomly here, the process might reasonably be called "*random-ary exponentiation*". Space and time efficiency were shown in [11] to be comparable with 4-ary exponentiation. For application to integer RSA, the multiplication is the operation in the multiplicative group of residues for the chosen modulus. Explicit mention of the modulus is not necessary. Termination is guaranteed because only base choices greater than 1 are allowed, and so D decreases on every iteration. Correctness is easily established using the loop invariant in terms of the initial value $D.Init$ of D.

The choices of base set and associated addition chains for each base/digit pair (m, d) are made with security and efficiency in mind. In particular, for efficiency the choice of addition chain for raising to the power m always includes d so that the computation of Q^m provides Q^d *en route* at little or no extra cost. Thus, these two power computations are not performed independently and consecutively, as might be implied by the code. They are to be implemented so that Q^m uses all the work done already to compute Q^d. So, in the case of RSA, the main cost of a loop iteration is only the cost of computing Q^m plus the conditional extra multiplication involving P.

The random choice of base values from a pre-chosen fixed set achieves different exponentiation schemes on successive runs and so makes impossible the usual averaging process required for differential power or electro-magnetic analysis (DPA/DEMA) [4,6].

Unlike the case for m-ary exponentiation, by reversing the direction of processing the exponent, *both* arguments in the conditional product are changed for every multiplication. In general, because the powers of C are always increasing, no power of C is repeatedly re-used during the exponentiation. So the attack described in [9] on a single exponentiation is inapplicable in its current form.

For convenience, the processing of the exponent D is presented as being performed within the main loop. For security reasons, it should probably be scheduled differently. The illustrated processing order may be less secure from the point of DPA or DEMA because it can reveal the random choice of the local base m, which should remain secret. Instead, the selection of the base and associated addition chain instructions can be performed by the CPU on-the-fly while the exponentiation is performed in parallel by a crypto co-processor, or it can be done in advance and stored when there is no co-processor. At any rate, these computations should be scheduled so as not to reveal the end points of each iteration of the main loop. Otherwise, the number and type of long integer operations during the loop iteration may leak enough information about the values of m and d, enabling D to be reconstructed. This paper shows how such data might be used to determine possible values for D.

3 The Base Choice and Addition Sub-Chains

A typical safe set of allowable bases is {2,3,5}. The full list of minimal addition sub-chains for these bases is given in Table 1. For example, the third case there corresponds to computing C^5 using the three multiplications $C^1 \times C^1 = C^2$, $C^1 \times C^2 = C^3$ and $C^2 \times C^3 = C^5$. The first three addition chains provide C^d when digit d is 0, 1, 2 or 3: for $0 < d < m$ the chain already contains the value of d, while the case $d = 0$ requires no multiplication and so 0 does not need to appear. The last addition chain can be used when $d = 4$. *Minimal* here means that any other addition chains which give a power equal to the base are longer. The subchains in Table 1 are minimal. To achieve the fastest exponentiation, longer chains are usually excluded, but they might improve security.

There is no instruction which updates the value of P in these addition subchains, but it can be represented explicitly using the following notation. Suppose the registers are numbered 1 for Q, 2 for *TempC* and 3 for P. Then the subchains can be stored as sequences of triples $(ijk) \in \{1,2,3\}^3$, where (ijk) means read

1+1=2	for base 2 with any digit d
1+1=2, 1+2=3	for base 3 with any digit d
1+1=2, 1+2=3, 2+3=5	for base 5 with any digit except 4
1+1=2, 2+2=4, 1+4=5	for base 5 with any digit except 3

Table 1. The Minimal Sub-Chains.

(m,d)	Multiplication Instructions
$(2,0)$	(111)
$(2,1)$	(112, 133)
$(3,0)$	(112, 121)
$(3,1)$	(112, 133, 121)
$(3,2)$	(112, 233, 121)
$(5,0)$	(112, 121, 121)
$(5,1)$	(112, 133, 121, 121)
$(5,2)$	(112, 233, 121, 121)
$(5,3)$	(112, 121, 133, 121)
$(5,4)$	(112, 222, 233, 121)

Table 2. A Choice for the Digit Sub-Chains.

(m,d)	0	1	2	3	4	p_m
2	0.3537	0.2757	-	-	-	$p_2 = 0.6294$
3	0.1826	0.0212	0.0244	-	-	$p_3 = 0.2283$
5	0.0936	0.0124	0.0110	0.0127	0.0126	$p_5 = 0.1423$

Table 3. ([11], Tables 6.2 and 6.3.) The limit probabilities $p_{m,d}$ of the base/digit pairs (m,d) and p_m for each base m.

the contents of registers i and j, multiply them together, and write the product into register k. In particular, P will always be updated using a triple of the form (i33) and 3 will not appear in triples otherwise. Now, adding in the instruction for updating P yields the list of subchains given in Table 2 as one possibility. It contains one representative for each base/digit pair (m,d). Other choices are possible. Such a table requires only a few bytes of storage.

The way in which the base is chosen from the allowable set has efficiency and security implications. In [11] it was shown that the following choice provided efficiency better than the binary method and nearly as good as the 4-ary method: (Here the function *Random* returns a fresh, random real in the range [0,1].)

```
m ← 0 ;
If Random < 7/8 then
    If D mod 2 = 0 then m ← 2 else
    If D mod 5 = 0 then m ← 5 else
    If D mod 3 = 0 then m ← 3 ;
If m = 0 then
Begin
    p ← Random ;
    If p < 6/8 then m ← 2 else
    If p < 7/8 then m ← 5 else
                    m ← 3 ;
End ;
```

The resulting probabilities p_m of each base m and $p_{m,d}$ of each base/digit pair (m, d) occurring in the representation of D are given in Table 3[2]. They will be used later to assess whether certain attacks are feasible. From them it is possible to work out the average number of bases used in a MIST exponentiation scheme:

Theorem 1. ([11], Thm. 7.2) *With the choices above, the average number of digits in a* MIST *representation for D is approximately*[2] $0.7566 \times \log_2 D$.

4 The Sequence of Addition Chain Values

We now turn away from the powers of C generated during an exponentiation and concentrate on the integers contained in the corresponding addition chain. The individual addition sub-chains for each base can be formed easily into an addition chain which describes a complete exponentiation scheme for D. In terms of the triples in Table 2, the sub-chain lists just need to be concatenated. Each value is associated with one of the variables Q, *TempC* or P according to the register in which the corresponding power of C is to be written. We will work with addition chains containing this extra detail. If S is the final value associated with Q at the end of one subchain, then, by applying the instructions listed in Table 2, the values computed in the next subchain are those listed in Table 1 multiplied by S, together with any which occurs for P.

Reconstruction of the sequence of digits and hence determination of the secret exponent is investigated using knowledge of which of these addition chain elements are equal, and which share equal summands. The following theorems will be used to show that, for the most part, we only need to look locally in the chain for such equality or sharing. With fairly minimal and reasonable restrictions on the choices of base set and associated addition sub-chains, these theorems hold much more generally.

Theorem 2. *The integers (i.e. exponents) associated with Q and P at the start of successive subchains form monotonically increasing sequences. That for Q is strictly increasing and strictly dominates that for P. At the start of each subchain, Q is associated with the largest integer in the addition chain up to that point.*

Proof. Initially Q is associated with 1 and the other registers with 0. So the domination property holds for the first values. Thereafter, suppose Q, *TempC* and P contain the S, T and Uth powers of C respectively at the start of a

[2] The figures here are corrected after a minor bug in the software for [11].

subchain for (m,d). Assume $T < S$ and $U < S$. The next values associated with Q and P are $m{\times}S$ and $U{+}d{\times}S$. Then $d < m$ means $U{+}d{\times}S < S{+}(m{-}1){\times}S = m{\times}S$. So the next value for Q is larger than the next for P. Hence the sequence for Q dominates that for P. Since $1 < m$, $S < m{\times}S$ and the next Q is larger than it was at the beginning of the subchain. So that sequence is strictly increasing. Moreover, as $U \leq U{+}d{\times}S$ the sequence for P is also increasing monotonically.

The exponents associated with *TempC* are multiples of S since the initial value T for *TempC* is unused, and the value U of P is only used to update P. So, as there are no other operations, these exponents only involve integers obtained *en route* from S to the next Q value, and are strictly smaller than it. Thus, at the start of each subchain, Q does indeed contain the largest power so far calculated. □

Theorem 3. *Except for initialisation and the calculation of the first non-trivial value for P, the addition chain contains no sum result more than once.*

Proof. There are two cases to consider. First, suppose some integer is recomputed in two different subchains. Assume the second of these subchains initially has integers S and U in Q and P respectively. Any updating operation in this subchain or any future subchain gives an integer which is a linear combination $\sigma S{+}\pi U$ for integers $\sigma > 0$ and $\pi \in \{0,1\}$. This only creates integers $\geq S$. Pick σ and π to give the duplicated value. By the previous theorem, updating operations in previous subchains only created integers at most S. Hence S is the recomputed value and so $\sigma S{+}\pi U = S$. Since $S > 0$ and $U \geq 0$, this can only hold if $\sigma \leq 1$. Multiplications have at least two arguments, so $\sigma{+}\pi \geq 2$. Hence $\sigma = \pi = 1$, from which $U = 0$. This solution occurs only when the digit d is 1 and P is updated to its first non-trivial value. So the value recomputed in the second subchain is uniquely determined. Moreover, since by the previous theorem the values in Q are strictly increasing and represent the largest integers so far calculated, the first computation of S is only in the immediately preceding subchain. S exceeds the initial value $U = 1$ in P. So S is calculated just once in the first of these two subchains.

Now consider the case where a value is recomputed within a single chain. Such recomputation must involve an updating of P because the operations which write to Q and *TempC* generate a strictly increasing sequence. We use the same notation again. The value of the updated P is $U{+}dS$ where $d > 0$ is the digit. All other subchain additions output an integer of the form σS for some integer $\sigma > 1$. So, choosing σ from another equal value gives $\sigma S = U{+}dS$ and hence $U = (\sigma{-}d)S$ where $0 \leq U < S$. This is impossible in integers unless $U = 0$ and $\sigma = d$. So P is being given its first non-trivial value, and dS is recomputed. We have $d > 1$ in this case since the computation of dS is done in the same subchain as P is updated. □

In fact, the re-computation of the same power of C should never occur: a good implementation should avoid the useless multiplication by 1, and should also avoid the apparent need to write the initial non-trivial value to P by using the value written to Q or *TempC* and renaming that register as P. The corresponding entries in the addition chain can then be omitted.

Example. Continuing with the example from Section 2, the sequences of exponents for Q and P are 1, 3, 6, 30, 60, 180, 360 and 0, 1, 25, 55, 235 respectively. The first non-zero value of P, namely 1, appears already in the sequence for Q, but there are no other repetitions.

5 Re-Use of Summands

Our main assumption in the first threat model is that the attacker can recognise the re-use of operands. Such re-use occurs when members of the addition chain share summands. So we need to know when this can happen, i.e. when two sums in the addition chain share a common input.

Theorem 4. i) *No integer different from the first non-zero value for P is used as a summand in more than three addition chain members. Addition chain elements which share such common summands all belong to the same digit subchain. They all lie within a sequence of at most four consecutive operations, and at most one of those with the shared summand is a doubling. If three sequential operations share such a common summand, then the digit associated with the subchain to which they all belong is non-zero.*

ii) *The first non-zero value for P may be used as a summand in up to four different addition chain operations. All but the last of the sums which use this value of P belong to the same subchain, while the last (which updates P to its second non-trivial value) belongs to a different subchain and may be arbitrarily many operations after the initial case.*

Proof. First consider the summands used in the sums on either side of a digit sub-chain boundary where the Q value is S. All operands below the boundary are less than S because S is the largest value computed up to that point, and it has not been used as an operand yet. Above the boundary, all operands are S or a multiple thereof with the single exception of the previous value for P when it is next updated. So, if there are two equal summands belonging to different digit sub-chains, they must be equal to a value of P.

However, from Theorem 3 we know that, apart from the first non-zero value, values of P are distinct from each other and from values in Q or *TempC*. So, as Q and *TempC* are computed from previous values of Q and *TempC*, their arguments cannot have values equal to those acquired by P unless those arguments are equal to the first non-zero value of P. Thus, with only that possible exception, each value of P is used at most once as a summand, namely in the next sum which updates its value. Consequently, two equal summands belonging to different digit sub-chains must actually be equal to the *first* non-zero value of P.

Hence, in case (i) for arguments different from the first non-zero value of P, equal summands appear within the same digit sub-chain. Thus they lie within a sequence equal to the longest such sub-chain, which is 4 here. Each doubling involves different operands (otherwise there would be no point in performing the subsequent doublings) and so the set of sums with a shared summand will contain at most one doubling. Checking through all the subchains given in Table 1, no operand is used in more than two operations. Moreover, such operations are adjacent and at most one of them is a square. So, when the sum which updates P is included in the middle of the sub-chain, at most three operations can occur using the same operand and, for our choice of insertion points, they are sequential. Of course, this only occurs when $d \neq 0$.

Now take case (ii) where the first non-trivial value for P is the operand under consideration. As with the other case, this operand value is used in at most two additions for updating Q and *TempC*. It is also used in at most two additions which update P. The first of these may have been optimised out, replacing the multiplication by 1 with an initialisation. This addition and those involving Q and *TempC* all occur in the same digit sub-chain, as before. The last use, namely the second to update P, occurs for a subsequent sub-chain after arbitrarily many bases for each of which the digit is 0. □

Example. Continuing with the example from Section 2, the first three pairs $(m, d) = (3, 1)$, (2,0) and (5,4) generate the instructions 112, 133, 121, 111, 112, 222, 233, 121. These produce:

$$\begin{array}{llllllll}
\mathit{TempC} & = C^1{\times}C^1 & = C^2 & ; \qquad & P & = C^1{\times}C^0 & = C^1 & ; \\
Q & = C^1{\times}C^2 & = C^3 & ; \qquad & Q & = C^3{\times}C^3 & = C^6 & ; \\
\mathit{TempC} & = C^6{\times}C^6 & = C^{12} & ; \qquad & \mathit{TempC} & = C^{12}{\times}C^{12} & = C^{24} & ; \\
P & = C^{24}{\times}C^1 & = C^{25} & ; \qquad & Q & = C^6{\times}C^{24} & = C^{30} & .
\end{array}$$

Operand C^1 is the first non-trivial value of P and it is used 4 times: the first three lie in the subchain for $(m, d) = (3, 1)$ and the last occurs in the subchain for $(m, d) = (5, 4)$. The intermediate subchain has $d = 0$.

6 Identifying the Digit Sub-Chains

In order to describe detailed operand sharing in a sequence of operations some further notation is needed. Let (123)(34) mean that in a list of exactly 4 operations, the first three share a common operand, the third and fourth share a different common operand, and no other operations in that list share a common operand. So the numbers in the cycles represent positions in the sequence of operations, starting at 1, and two operations will share a common operand if, and only if, their position numbers both belong to a common cycle in the list. Since a square or a doubling shares an operand with itself, the number of each square or doubling appears twice in its cycle, as in (112). Also, an operation which does not share operands with any other operation (or itself) will appear in a cycle on its own, as in (2). However, there are no cases of this here. With this notation, the subchains listed in Table 2 share operands as in Table 4.

Now assume that the operand sharing pattern is known for the complete addition chain. By Theorem 3, except for the use of the first non-trivial value of P, operands which are equal in the addition chain are equal because they were

(m, d)	Operand Sharing
$(2, 0)$	(11)
$(2, 1)$	(112)
$(3, 0)$	(112)
$(3, 1)$	(1123)
$(3, 2)$	(113)(23)
$(5, 0)$	(112)(23)
$(5, 1)$	(1123)(34)
$(5, 2)$	(113)(234)
$(5, 3)$	(112)(24)(34)
$(5, 4)$	(114)(22)(34)

Table 4. Operand Sharing within each Digit Sub-Chain.

explicitly selected equal in the sub-chain construction. By Theorem 4, we know that the last use of the exceptional value of P as an operand is to update P, and so it is not a square. Hence the squares in the addition chain are exactly those expected from the structure of the component sub-chains.

It is now mostly straightforward to deduce what the individual subchains are, and hence the sequence of bases and digits: each square (doubling) denotes the start of a new sub-chain with the exception of those which are the second operation in a subchain for $(m, d) = (5, 4)$. When this case occurs, there is operand sharing between the first and fourth operations of the sub-chain. This is expressed in the pattern (114). According to Theorem 4, there is normally no sharing of operands between different sub-chains. Hence, when the pattern (114) is not observed, we know that normally both squares mark the start of different sub-chains.

The only possible exception is if the shared operand in (114) is the first non-zero value taken by P. Then the sharing pattern (114)(22)(34) for (5,4) must ambiguously represent two division sub-chains which have lengths 1 and 3 respectively. The second subchain has pattern (11)(23) (when the operations are re-numbered from 1 to 3). However, that pattern does not correspond to any occurring in Table 4. So this case cannot arise. Hence:

Theorem 5. *The pattern of operand re-use in an addition chain determines the boundaries of each digit sub-chain uniquely.*

In practice, this partitioning is performed by identifying the doublings first and then writing down the patterns for operand sharing between the operations within each partition. If the pattern (11)(23) emerges, then its partition needs to be merged with the previous one.

Example. In the same example as before, the operand sharing pattern is (11237)(44)(558)(66)(78)(9 9 10)(11 11 12)(13 13 14). Partitioning this before each square yields (11237); (44); (558); (66)(78); (9 9 10); (11 11 12); (13 13 14). Re-numbering to make each sub-chain start with instruction 1 gives (11237); (11); (114); (11)(23);

$(112); (112); (112)$. As 7 exceeds the length of the longest sub-chain, it must represent the updating of P to its second non-trivial value. So we delete it to obtain sharing only within sub-chains. Also, we must merge (114) and (11)(23) since the latter is not a pattern in Table 4. This produces $(1123); (11); (114)(22)$ $(34); (112); (112); (112)$ from which we can extract possible choices for the pairs (m, d): first (3,1); (2,0); (5,4) and then three occurrences of $(2, 1)$ or $(3, 0)$.

It is evident from Table 4 that every (m, d) has a distinct pattern of operand sharing except for two: $(2, 1)$ and $(3, 0)$ have identical patterns (112). Thus operand sharing almost determines the sequence of pairs (m, d):

Theorem 6. *The pattern of operand re-use in an addition chain determines the sequence of pairs* (m, d) *up to an ambiguity between* $(2, 1)$ *and* $(3, 0)$.

Theorem 7. *The average number of exponents with addition chains that have the same operand sharing pattern as one for* D *is at least* $D^{1/3}$.

Proof. By Theorem 6, base/digit pairs can be derived in most cases. The only ambiguities occur between the cases $(2, 1)$ and $(3, 0)$. Assuming successive base choices are independent, then the probability of an ambiguous case is $p_{2,1}+p_{3,0} = 0.4583$. Hence almost every other subchain has two possible choices for the base/digit pair. By Theorem 1, an average exponent contains $0.7566\times \log_2 D$ subchains. Hence the expected number of different matching exponents is about $2^{0.7566\times \log_2 D\times 0.4583} = D^{0.7566\times 0.4583} = D^{0.347}$ □

Remarks. i) The choice of base is not constrained here. A deterministic selection of, for example, a base which exactly divides the current value of D would place structural constraints which would further reduce the possible choices for D.
ii) Successive base choices are not independent, but this makes only a marginal difference to the exponent 0.347.
iii) We have assumed different choices of (m, d) lead to different values for D. This is almost always true, and makes no practical difference to the number of exponents that need to be considered in an attack.

Example. In the previous example, operand sharing gave 8 choices for the division chain. The first three (m, d)s yield $(Q, P) = (C^{30}, C^{25})$. The next (2,1) or (3,0) produces (C^{60}, C^{55}) or (C^{90}, C^{25}). The following (2,1) or (3,0) doubles the possibilities to (C^{120}, C^{115}), (C^{180}, C^{115}), (C^{180}, C^{55}) or (C^{270}, C^{25}). The last alternative theoretically doubles this number, but (3,0) is impossible because the final (top) digit d must be non-zero. So, applying (2,1), the final output P is one of C^{235}, C^{295}, C^{235} or C^{295}. The example captures one of the few sources of repeated values, which arises from the property 2+1 = 3+0.

7 The Operand Sharing Search Space

Let us assume that the Mist algorithm is being used because re-use of operands can be detected, thereby making the standard algorithms unsuitable. So the main assumption in the threat model here is that identical operands can be detected, perhaps by averaging the power traces of digit-by-digit products using the method given in [9], or by observing different addresses being sent along the internal bus.

It is straightforward to show that, with a negligibly small number of exceptions, operand sharing in the addition chain occurs if, and only if, operand sharing occurs in the exponentiation. This is because $C^i = C^j \Rightarrow i = j$ holds almost always. Since there are only some $0.7566 \log_2 D$ sub-chains in which to check operand sharing, exceptions are unlikely to occur, even in a complete addition chain. Then Theorems 6 and 7 provide:

Theorem 8. *If an attacker can determine operand re-use from side channel leakage, then he can almost certainly deduce the sequence of pairs* (m, d) *used in the* MIST *exponentiation scheme up to an ambiguity between* $(2, 1)$ *and* $(3, 0)$. *This reduces the search space for the correct exponent to about* $D^{1/3}$.

The known ratio $p_{3,0} : p_{2,1} = 0.1826 : 0.2757$ enables the choices for D to be ranked with those closest to this ratio being selected first. Then on average fewer exponents would need to be investigated before the correct one is determined. However, for a 384-bit exponent, say, the number of choices given by the theorem is still around 2^{133} and that for a 512-bit exponent is around 2^{177}. These are reasonable minimum choices for when the Chinese Remainder Theorem is or is not used. For RSA, both cases are computationally infeasible for the foreseeable future. However, for ECC a typical 192-bit key would really be unsafe. Of course, key lengths can still be increased for both of these if necessary.

In the case of the m-ary method, knowledge of operand sharing enables the exponent to be deduced immediately without any further calculations [9]. So the MIST algorithm is much stronger against such an attack.

8 S&M Chains

A much weaker threat model is that the attacker can distinguish between squares (S) and multiplies (M). The first main task he has is to parse correctly the word created from the alphabet $\{S, M\}$ which is generated by the operations of the exponentiation scheme. We will call the word an *S&M chain*. It must correspond to a division chain [8]. The patterns of the S&M subchains corresponding to each pair (m, d) are listed in Table 5 and their probabilities in Table 6.

Theorem 9. *Suppose squares and multiplies can be distinguished, but not individual reuse of operands. Then the average number of exponents which can generate the same sequence of squares and multiplications as a given one for D is bounded below by* $D^{3/5}$.

Proof. As before, the occurrences of S determine almost all of the subchain boundaries exactly. The exception is the case $(5, 4)$ for which *SSMM* may split as *S* and *SMM*. Suppose we perform this split. Then the number of subchains is increased by a factor of $1+p_{SSMM}$ and the probabilities of the minimal such S&M sequences are then:

(2,0)	*S*	(5,0)	*SMM*
(2,1)	*SM*	(5,1)	*SMMM*
(3,0)	*SM*	(5,2)	*SMMM*
(3,1)	*SMM*	(5,3)	*SMMM*
(3,2)	*SMM*	(5,4)	*SSMM*

Table 5. The S&M Sub-Chains for each Pair (m, d).

$$\begin{array}{lcll} p_S & = & p_{2,0} & = 0.3537 \\ p_{SM} & = & p_{2,1} + p_{3,0} & = 0.4583 \\ p_{SMM} & = & p_{3,1} + p_{3,2} + p_{5,0} & = 0.1393 \\ p_{SMMM} & = & p_{5,1} + p_{5,2} + p_{5,3} & = 0.0361 \\ p_{SSMM} & = & p_{5,4} & = 0.0126 \end{array}$$

Table 6. The S&M Sub-Chain Probabilities.

$$\begin{array}{lcll} p'_S & = & (p_S{+}p_{SSMM})/(1{+}p_{SSMM}) & = 0.3618 \\ p'_{SM} & = & p_{SM}/(1{+}p_{SSMM}) & = 0.4526 \\ p'_{SMM} & = & (p_{SMM}{+}p_{SSMM})/(1{+}p_{SSMM}) & = 0.1500 \\ p'_{SMMM} & = & p_{SMMM}/(1{+}p_{SSMM}) & = 0.0356 \end{array}$$

Assume, for simplicity, that the successive choices of base are independent[3]. Then the number of choices for the base/digit pair underlying *SMMM* is 3, that for *SM* is 2, that for *S* is 1. If *SMM* is preceded by *M*, it corresponds to a complete subchain and there are 3 choices for it. However, if *SMM* is preceded by *S*, the two can be merged to form *SSMM*, giving 4 choices. Taking into account the proportion of *SMM*s derived from *SSMM*, p'_{SMM} breaks up into $p'_{SSMM} = p'_{SMM}(p_{SSMM}{+}p_Sp_{SMM})/(p_{SMM}{+}p_{SSMM}) = 0.0611$ for the latter case and $p'_{MSMM} = p'_{SMM}{-}p'_{SSMM} = 0.0889$ for the former. Then the average number of ways of selecting a base/digit pair from an S&M subsequence (including repartitioning *SSMM*) is $1^{p'_S} \times 2^{p'_{SM}} \times 3^{p'_{SMMM}+p'_{MSMM}} \times 4^{p'_{SSMM}} = 1.7079$.

An average exponent contains $0.7566 \log_2 D$ subchains, and $1{+}p_{SSMM}$ times more minimal S&M subsequences. Hence the expected number of different matching exponents is about $1.7079^{(1+p_{SSMM})\times 0.7566 \log_2 D} = D^k$ where $k = (1{+}p_{SSMM}) \times 0.7566 \log_2(1.7079) \approx 0.5916 \approx \frac{3}{5}$. □

Even more so than in the case of known operand sharing, the above demonstrates that recovery of D is well outside the reach of an attacker when only the sequence of squares and multiplies leaks for individual exponentiations.

Clearly, different choices of bases and addition sub-chains will provide different probabilities and hence may increase or decrease the strength of a given attack. Care is therefore necessary in these choices. For example, the present choices have been consciously selected to make the probabilities $p_{2,1}$ and $p_{3,0}$

[3] Overall, *SMM* occurs with probability 0.1393, but after *S* its probability is 0.1410.

the highest after $p_{2,0}$, which itself needs to be high to provide the requisite efficiency. This decreases the effectiveness of the above attacks. Moreover, the choice of sub-chains means there are no long S&M sequences with a unique base/digit interpretation which would prune the search space for matching exponents down to a computationally feasible proposition.

9 Conclusion

The "Mist" randomary exponentiation algorithm has a variety of features which make it much more resilient to attack by differential power or electro-magnetic analysis than the normal m-ary or sliding window methods. Mist uses randomly different multiplication schemes on every run in order to avoid the averaging which is normally required for such side channel attacks to succeed.

The algorithm also avoids wide re-use of multiplicands within a single exponentiation, thereby defeating some other potentially more powerful attacks. Knowledge of such operand re-use reduces the search space to about $D^{1/3}$ for an exponent D, but this leaves an infeasible quantity of computing for standard RSA applications. Furthermore, knowledge of only the sequence of squares and multiplies reduces the search space to around $D^{3/5}$. These search spaces might be reduced if an effective way can be found to share data deduced from different exponentiations. This is an open problem, but the possible danger should be adequately protected against by standard exponent blinding.

In consequence, the algorithm appears to be much safer than the standard binary, m-ary or sliding windows techniques against current state-of-the-art in DPA and DEMA side channel attacks, yet it makes no greater use of either space or time.

References

1. K. Gandolfi, C. Mourtel & F. Olivier, *Electromagnetic Analysis: Concrete Results*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 251–261.
2. D. E. Knuth, *The Art of Computer Programming*, vol. **2**, "Seminumerical Algorithms", 2nd Edition, Addison-Wesley, 1981, 441–466.
3. P. Kocher, *Timing Attack on Implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology – Crypto '96, N. Koblitz (editor), Lecture Notes in Computer Science, **1109**, Springer-Verlag, 1996, 104–113.
4. P. Kocher, J. Jaffe & B. Jun, *Differential Power Analysis*, Advances in Cryptology – Crypto '99, M. Wiener (editor), Lecture Notes in Computer Science, **1666**, Springer-Verlag, 1999, 388–397.
5. E. Oswald & M. Aigner, *Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 39–50.

6. J.-J. Quisquater & D. Samyde, *ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards*, Smart Card Programming and Security (E-smart 2001), Lecture Notes in Computer Science, **2140**, Springer-Verlag, 2001, 200–210.
7. J.-J. Quisquater & D. Samyde, *Eddy current for Magnetic Analysis with Active Sensor*, Smart Card Programming and Security (E-smart 2002), Lecture Notes in Computer Science, Springer-Verlag, 2002 *to appear.*
8. C. D. Walter, *Exponentiation using Division Chains*, IEEE Transactions on Computers, **47**, No. 7, July 1998, 757–765.
9. C. D. Walter, *Sliding Windows succumbs to Big Mac Attack*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 286–299.
10. C. D. Walter, *Improvements in, and relating to, Cryptographic Methods and Apparatus*, UK Patent Application 0126317.7, Comodo Research Laboratory, 2001.
11. C. D. Walter, *MIST: An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis*, Topics in Cryptology – CT-RSA 2002, B. Preneel (editor), Lecture Notes in Computer Science, **2271**, Springer-Verlag, 2002, 53–66.