Some Security Aspects of the Mist Randomized Exponentiation Algorithm

Colin D. Walter*

Comodo Research Laboratory
10 Hey Street, Bradford, BD7 1DQ, UK
www.comodo.net

Abstract. The Mist algorithm is used in embedded crypto-systems to provide protection against power analysis attacks. It generates randomly different addition chains for performing a particular exponentiation. This means that power attacks which require averaging over a number of exponentiation power traces becomes 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 expoent search space might be considerably reduced, the number of possible exponents is still well outside the range of feasible computation in the foreseeable future.

Key words: Mist exponentiation algorithm, division chains, addition chains, power analysis, DPA, blinding, smart card.

1 Introduction

Recent progress in side channel attacks [4], [5] on embedded cryptographic systems has exposed 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 and Diffie-Hellman. Initial power attacks required averaging over a number of exponentiations [5]. 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 operation very clearly. Fortunately, such attacks can be defeated by modifying the exponent e to $e+rg$ where r is a random number and g is the order of the (multiplicative) group in which the exponentiation is performed [4]. This results in a different exponentiation procedure being performed every time.

* Work done while the author was at the Computation Department, UMIST, Manchester, UK

However, the author showed recently [11] that there were strong theoretical grounds for believing that, given the right monitoring equipment [2], it would be possible to break the normal m-ary exponentiation method [3] and related sliding windows techniques using a single exponentiation. This method relies on being able to recognise the same operands being reused over and over, namely the pre-computed powers of the initial text, and requires no knowledge of the modulus, input text or output text. It renders useless the choice of $e+rg$ 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 [8] have given one such example. For integer RSA, a novel exponentiation algorithm called "Mist" was presented at RSA 2002 [13]. This seems to avoid all of the above-mentioned pitfalls. It is more time efficient than the standard binary method, is comparable in space usage to 4-ary exponentiation, and can also be combined with any counter-measures which modify the arguments. The algorithm relies on the generation of random addition chains [3] which determine the operations to be performed, and is based on previous work by the author [9] for finding efficient exponentiation schemes.

The Mist algorithm was created to defeat power anaylsis attacks which are able to detect the re-use of arguments. The main aim here is to establish that such knowledge does not significantly reduce the effectiveness of the algorithm. Although operand re-use provides a handle which prunes a search tree for exponents considerably, it is still computationally infeasible to attempt to break the crypto-system in this way. A more likely scenario is that the attacker can only distinguish squares from multiplies. Such data also prunes the search space considerably, but again leaves an infeasible quantity of computing.

2 The Mist Algorithm

For notation, let us assume that M^E has to be computed. D will always represent a "divisor" in the sense of [9], and R a residue modulo D. A set of divisors D is chosen in advance (it will be $\{2, 3, 5\}$ here), and an associated table of addition chains for raising to the power D is stored in memory. Several variables are used: there are three for long integers which contain powers of M, namely $StartM$, $TempM$ and $ResultM$. Of these, $TempM$ is for temporary storage when $StartM$ is raised to the power D, and so does not occur explicitly in the following code. E contains the power to which $StartM$ still has to be raised before the exponentiation is complete.

The Mist Exponentiation Algorithm

{ Pre-condition: $E \geq 0$ }
```
StartM  := M  ;
ResultM := 1  ;
While E > 0 do
Begin
   Choose a random "divisor" D ;
   R := E mod D ;
   If R ≠ 0 then
      ResultM := StartM^R×ResultM ;
   StartM := StartM^D ;
   E      := E div D ;
   { Loop invariant: M^(E.Init) = StartM^E×ResultM }
End ;
{ Post-condition: ResultM = M^(E.Init) }
```

(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 divisors greater than 1 are allowed, and so E decreases on every iteration. Correctness is easily established using the loop invariant in terms of the initial value $E.Init$ of E.

The random choice of divisors achieves different exponentiation schemes on successive runs and so makes impossible the usual averaging process required for differential power analysis [5].

When the divisor set consists of the single divisor 2, the algorithm simplifies to the right-to-left binary exponentiation algorithm, that is, to the standard version of square-and-multiply in which the least significant exponent bit is processed first. However, unlike the left-to-right version of square-and-multiply, both arguments in the conditional product are changed for every multiplication. In general, because the powers of M are always increasing, no power of M is repeatedly re-used during the exponentiation. So the attack described in [11] on a single exponentiation is inapplicable in its current form.

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

For convenience, the processing of the exponent E is presented as being performed within the main loop. For security reasons, it *must* be scheduled differently. The illustrated processing order is insecure from the point of differential power analysis because it can reveal the random choice of the divisor D, which *must* remain secret. The selection of the divisor and associated addition chain instructions can be performed by the CPU on-the-fly while the exponentiation is performed in parallel by the co-processor, or it can be done in advance 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 the values of D and R, enabling E to be reconstructed.

3 The Divisor Choice and Addition Sub-Chains

A typical safe set of divisors is {2,3,5} [13]. The full list of minimal addition sub-chains for the divisors is given in Figure 1. The third case there corresponds

1+1=2	for divisor 2 with any residue R
1+1=2, 1+2=3	for divisor 3 with any residue R
1+1=2, 1+2=3, 2+3=5	for divisor 5 with any residue except 4
1+1=2, 2+2=4, 1+4=5	for divisor 5 with any residue except 3

Fig. 1. The Minimal Sub-Chains.

to using an initial M^1 to compute M^2 with one multiplication, then M^3 with another multiplication, and finally M^5 with a third multiplication. The first three addition chains provide M^R when R is 0, 1, 2 or 3: for $0 < R < D$ the chain already contains the value of R, while the case $R = 0$ requires no multiplication and so 0 does not need to appear. The last addition chain can be used when $R = 4$. *Minimal* here means that any other addition chains which give a power equal to the divisor are longer. The subchains above 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 $ResultM$ in the above addition subchains, but it can be represented explicitly using the following notation. Suppose we number the registers 1 for $StartM$, 2 for $TempM$ and 3 for $ResultM$. Then the subchains can be stored as sequences of triples $(ijk) \in \{1, 2, 3\}^3$, where (ijk) means read the contents from registers i and j, multiply them together, and write the product into register k. In particular, $ResultM$ 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 $ResultM$ yields the list of subchains given in Figure 2 as one possibility. It contains one representative for each divisor/residue pair (D, R). Other choices are possible. Such a table requires only a few bytes of storage.

The way in which the divisor is chosen from the allowable set has efficiency and security implications. In [13] it was shown that the following choice provided efficiency better than the binary method and nearly as good as the 4-ary method: (Here *Random* returns a fresh random real in the range [0,1].)

(D, R)	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)

Fig. 2. A Choice for the Divisor Sub-Chains.

```
D := 0 ;
If Random(x) < 7/8 then
    If E mod 2 = 0 then D := 2 else
    If E mod 5 = 0 then D := 5 else
    If E mod 3 = 0 then D := 3 ;
If D = 0 then
Begin
    p := Random(x) ;
    If p < 6/8 then D := 2 else
    If p < 7/8 then D := 3 else
                    D := 5 ;
End ;
```

The probabilities p_D of each divisor D and $p_{D,R}$ of each divisor/residue pair (D, R) occurring in the division chain are given in Tables 3 and 4 [13]. They will be used later to assess whether or not the attacks are feasible. From them it is possible to work out the average number of divisors used in a MIST exponentiation scheme [13]:

Theorem 1. *The average number of subchains in a* MIST *addition chain for E is approximately* $0.753 \times \log_2 E$.

(D, R)	0	1	2	3	4
2	0.350	0.271	-	-	-
3	0.189	0.0209	0.0242	-	-
5	0.0979	0.0120	0.0106	0.0123	0.123

Fig. 3. The limit probabilities $p_{D,R}$ of the divisor/residue pairs (D, R).

$$p_2 = 0.621$$
$$p_3 = 0.234$$
$$p_5 = 0.145$$

Fig. 4. The limit probability p_D for each divisor D.

4 The Sequence of Addition Chain Values

We now turn away from the powers of M generated during an exponentiation and concentrate on the integers contained in the corresponding addition chain. The individual addition sub-chains for each divisor can be formed easily into an addition chain which describes a complete exponentiation scheme for E. 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 $StartM$, $TempM$ or $ResultM$ according to the register in which the corresponding power of M is to be written. We will work with addition chains containing this extra detail. If S is the final value associated with $StartM$ at the end of one subchain, then, by applying the instructions listed in Table 2, the values computed in the next subchain are just those listed in Figure 1 multiplied by S, together with any which occurs for $ResultM$. Reconstruction of the sequence of divisors and hence determination of the secret exponent is performed 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 divisor set and associated addition sub-chains, these theorems hold much more generally.

Theorem 2. *The integers associated with $StartM$ and $ResultM$ at the start of successive subchains form monotonically increasing sequences. That for $StartM$ is strictly increasing and strictly dominates that for $ResultM$. At the start of each subchain, $StartM$ contains the largest integer in the addition chain up to that point.*

Proof. Initially $StartM$ is associated with 1 and the other registers with 0. So the domination property holds for the first values. Thereafter, suppose $StartM$, $TempM$ and $ResultM$ contain S, T and P respectively at the start of a subchain for (D, R) and both $T < S$ and $P < S$. The next values of $StartM$ and $ResultM$ will contain $D{\times}S$ and $P{+}R{\times}S$. Then $R < D$ means $P{+}R{\times}S < S{+}(D{-}1){\times}S = D{\times}S$. So the next value of $StartM$ is larger than the next $ResultM$. Hence the sequence for $StartM$ dominates the one for $ResultM$. Since $1 < D$, $S < D{\times}S$ and the next $StartM$ is larger than $StartM$ was at the beginning of the subchain. So that sequence is strictly increasing. Moreover, as $P \leq P + R{\times}S$ the sequence for $ResultM$ is also increasing monotonically.

The next integer in $TempM$ is a multiple of S since the initial value T in $TempM$ is unused, and the value P of $ResultM$ is only used to update $ResultM$.

So, as there are no other operations, the next $TempM$ must contain an integer obtained *en route* from S to the next $StartM$ value, and be strictly smaller than it. So at the start of each subchain, $StartM$ does indeed contain the largest integer so far calculated. □

Theorem 3. *Except for initialisation of the three variables to contain* 0 *and the calculation of the first non-trivial value for* $ResultM$*, 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 P in $StartM$ and $ResultM$ respectively. Any updating operation in this subchain or any future subchain gives an integer which is a linear combination $\sigma S{+}\pi P$ 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 $\sigma S{+}\pi P = S$. Since $S > 0$ and $P \geq 0$, 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 $P = 0$. This solution occurs only when the residue R is 1 and $ResultM$ 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 $StartM$ 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 $P = 1$ in $ResultM$. So S is calculated just once in the first of the two subchains.

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

In fact, the re-computation of the same power of M should never occur. A good implementation should avoid the useless multiplication by 1, and should also avoid the apparent need to write to $ResultM$ by using the value written to $StartM$ or $TempM$. The corresponding entries in the addition chain can then be omitted.

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* $ResultM$ *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 divisor subchain. They all lie within a sequence of at most four successive 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 residue associated with the subchain to which they all belong is non-zero.*
ii) *The first non-zero value for* $ResultM$ *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* $ResultM$ *belong to the same subchain, while the last 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 divisor sub-chain boundary where the $StartM$ 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 $ResultM$ when it is next updated. So, if there are two equal summands belonging to different divisor sub-chains, they must be equal to a value of $ResultM$.

However, from Theorem 3 we know that, apart from the first non-zero value, values of $ResultM$ are distinct from each other and from values in $StartM$ or $TempM$. So only the sums which update $ResultM$ use a value of $ResultM$ as an operand, other than the sum which yields the second non-zero value for $ResultM$. With the possible exception of the first non-zero value, each value of $ResultM$ is used only once as a summand, namely in the next sum which updates its value. So two equal summands belonging to different divisor sub-chains must actually be equal to the first non-zero value of $ResultM$.

Hence, in case (i), equal summands appear within the same divisor 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 Figure 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 $ResultM$ is included in the middle of the sub-chain, at most three operations can occur using the same operand and they must be sequential.

Now take case (ii) where the first non-trivial value for $ResultM$ is the operand under consideration. Unless the corresponding sum has been eliminated, it is used as an operand when $ResultM$ is updated to its first non-zero value and also when it is next updated. The first of these occurs in a subchain which, as in case (i), may use the operand for up to two other operations, making a total of up to four uses. The last use, namely the second to update $ResultM$, occurs after arbitrarily many consecutive divisors for all of which the residue is 0. □

6 Identifying the 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 3 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. With this notation, the subchains listed in Figure 2 share operands as in Table 5.

(D, R)	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)

Fig. 5. Operand Sharing within each Sub-Chain.

Now assume that the operand sharing pattern is known for the complete addition chain. It is mostly straightforward to deduce what the individual subchains are, and hence the sequence of divisors and residues. Each square (doubling) denotes the start of a new sub-chain with the exception of those appearing as the second operation in a subchain where $(D, R) = (5, 4)$. With one possible exception, this second operation cannot mark the beginning of a new sub-chain because of operand sharing expressed in the pattern (114) between the previous (first) operation and the next but one (fourth) operation. To be precise, according to Theorem 4, there is no sharing of operands between different sub-chains unless the shared operand is the first non-zero value taken by $ResultM$.

However, suppose the sharing pattern (114)(22)(34) does represent two division sub-chains. The first two sums being squares (doublings) ensures the subchains must have lengths 1 and 3 respectively. and the shared operand expressed in the cycle (114) must be the first non-trivial value taken by $ResultM$. Ignoring that sharing, the second subchain has pattern (11)(23) (when the operations are re-numbered from 1 to 3). However, that pattern does not corresond to any occurring in Table 5. Indeed, for any odd divisor, the first squaring (doubling) must share a summand with a later operation, and for the even divisor 2 the sub-chain has fewer than three operations. So we obtain a contradiction. Hence:

Theorem 5. *If the pattern of operand re-use is known in an otherwise unknown addition chain, then the boundaries of each divisor sub-chain can be determined successfully.*

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.

It is evident from Table 5 that every (D, R) 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 (D, R):

Theorem 6. *If the pattern of operand re-use is known in an otherwise unknown addition chain, then the sequence of pairs (D, R) can be deduced up to an ambiguity between $(2, 1)$ and $(3, 0)$.*

Theorem 7. *If operand sharing can be determined for the operations of an otherwise unknown addition chain for E, then the average number of exponents which match the determined patterns is above $E^{1/3}$.*

Proof. Divisor/residue pairs can be derived in most cases according to the theorem. The ambiguities only occur between the cases of $(2, 1)$ and of $(3, 0)$. Assuming successive divisor choices are independent, then the probability of an ambiguous case is $p_{2,1}+p_{3,0} = 0.4597$. Hence roughly every third subchain has two possible choices for the divisor/residue pair. By Theorem 1, an average exponent contains $0.753\times\log_2 E$ subchains. Hence the expected number of different matching exponents is about $2^{0.753\times\log_2 E\times0.4597} = E^{0.753\times0.4597} = E^{0.346}$ □

Remark. The choice of divisor is not constrained here. A deterministic selection of, for example, a divisor which exactly divides the current value of E would place structural constraints which would further reduce the possible choices for E.

7 A Power Analysis Attack

Let us assume that the MIST algorithm is being used because re-use of operands can be detected. So the main assumption in the threat model here is that identical operands can be detected from averaging the power traces of digit-by-digit products using, for example, the method given in [11].

For the attack described below, we now show that for any pair of RSA operands M^i and M^j which need to be compared, the property $M^i = M^j \Rightarrow i =$

j will also hold almost always. This will enable the attacker to deduce exactly which elements of the addition chain are equal from studying the power trace. A similar argument holds for other expected cryptographic applications.

In integer RSA, with probability almost equal to 1, the input text M is prime to the modulus $N = PQ$. Such values M form a group of order $\phi(N)$ which is the direct product of two cyclic groups of order $\phi(P)$ and $\phi(Q)$. For M^i and M^j to be equal, we need M to have order dividing $i-j$ in this group, it is necessary for it to be the product of the two generators raised to powers which are multiples of $\phi(P)/\gcd(P,i-j)$ and $\phi(Q)/\gcd(Q,i-j)$ respectively. So exactly $\gcd(P,i-j)\times\gcd(Q,i-j)$ elements will have order dividing $i-j$. Thus it will be unlikely that M is one of these elements if $i-j$ or the gcds are small: the probability is $\gcd(P,i-j)\times\gcd(Q,i-j)/(\phi(P)\phi(Q))$.

Here, we only have to compare powers of M which are used over four consecutive operations which start at the beginning of a sub-chain. This information is then applied to find out which prefix of the four operations has an operand sharing pattern which matches one of the subchains in Table 5. With the exception of previous values for $ResultM$, all operands are powers of the value M^S used in the first operation, which is a square. The largest power which can be formed for use in the last operation is then M^{8S}. The difference between the exponents i and j of two operands therefore has the form kS where $0 \le |k| < 8$. S itself is of the form $2^r 3^s 5^t$ for some potentially large r, s, t. However, the limited divisibility of $\phi(P)$ and $\phi(Q)$ by 2, 3, and 5 means the probability $\gcd(P,i-j)\times\gcd(Q,i-j)/(\phi(P)\phi(Q))$ of having a value of M which makes M^i and M^j equal is very small.

If one of the operands, M^i say, is a previous value of $ResultM$, then the randomness of i makes $i-j$ random and M^{i-j} is a random element of the group, with a probability of only $\phi(N)^{-1}$ of being the identity. Thus, again, the different operands are unlikely to be equal.

Since there are only some $0.753 \log_2 E$ sub-chains in which to check operand sharing, Theorem 6 and the methods used to derive it provide the next results:

Theorem 8. *If an attacker can determine operand re-use from side channel leakage, then he can almost certainly deduce the sequence of pairs (D, R) used in the* MIST *exponentiation scheme up to an ambiguity between $(2, 1)$ and $(3, 0)$.*

Corollary 9. *If an attacker can determine operand re-use from side channel leakage, then he can reduce the search space for the correct exponent to about $E^{1/3}$.*

The known ratio $p_{3,0} : p_{2,1} = 0.189 : 0.271$ enables the choices for E 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 was 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. Both cases are computationally infeasible for the foreseeable future. Of course, exponent length can still be increased 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 [11]. 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 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 call the word an *S&M chain*. The patterns of the S&M subchains are listed in Figure 6 and their probabilities in Figure 7

(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

Fig. 6. The S&M Sub-Chains for each pair (D, R).

$$\begin{aligned}
p_S &= p_{2,0} &&= 0.3501 \\
p_{SM} &= p_{2,1} + p_{3,0} &&= 0.4597 \\
p_{SMM} &= p_{3,1} + p_{3,2} + p_{5,0} &&= 0.1430 \\
p_{SMMM} &= p_{5,1} + p_{5,2} + p_{5,3} &&= 0.0349 \\
p_{SSMM} &= p_{5,4} &&= 0.0123
\end{aligned}$$

Fig. 7. The S&M Sub-Chains Probabilities.

Theorem 10. *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 E is bounded below by $E^{6/10}$.*

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 S&M sequences are then:

$$\begin{aligned}
p'_S &= (p_S + p_{SSMM})/(1+p_{SSMM}) &&= 0.358014 \\
p'_{SM} &= p_{SM}/(1+p_{SSMM}) &&= 0.454105 \\
p'_{SMM} &= (p_{SMM} + p_{SSMM})/(1+p_{SSMM}) &&= 0.153396 \\
p'_{SMMM} &= p_{SMMM}/(1+p_{SSMM}) &&= 0.034485
\end{aligned}$$

Assume, for simplicity, that the successive choices of divisor are independent. Then the number of choices for the divisor/residue pair behind $SMMM$ is 3, that for SM is 2, that for S is 1. If SMM corresponds to a complete subchain then there are 3 choices for it. However, the probability of SMM being preceded by S is p'_S. Hence, on average, we have $3+p'_S$ choices for the way SMM is allocated to a divisor/residue pair. Then the average number of ways of selecting a divisor/residue pair from an S&M subsequence (including repartitioning $SMMM$) is $1^{p'_S}\times 2^{p'_{SM}}\times(3+p'_S)^{p'_{SMM}}\times 3^{p'_{SMMM}} = 1.7134$.

An average exponent contains $0.753\log_2 E$ subchains, and $1+p_{SSMM}$ times more minimal S&M subsequences. Hence the expected number of different matching exponents is about $1.7134^{(1+p_{SSMM})\times 0.753\log_2 E} = E^k$ where $k = (1+p_{SSMM}) \times 0.753\log_2(1.7134) \approx 0.5920$. □

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

Clearly, different choices of divisors 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 $p_{3,0}$ the highest after $p_{2,0}$, which itself needs to be high to provide the requisite efficiency. This decreases the effectiveness of an attack using operand sharing. Moreover, the choice of sub-chains means there are no long S&M sequences with a unique divisor/residue intepretation which would prune the search space for matching exponents down to a computationally feasible proposition.

9 Conclusion

An exponentiation algorithm "MIST" has been presented which has a variety of features which make it much more resilient to attack by differential power 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 power analysis attacks to succeed.

The algorithm also avoids wide re-use of multiplicands within a single exponentiation, thereby defeating some of the more recent power analysis attacks. Knowledge of such re-use was seen to reduce the search space to about $E^{1/3}$ for an exponent E, but this leaves an infeasible quantity of computing. Furthermore, knowledge of the sequence of squares and multiplies only reduced the search space to around $E^{6/10}$.

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 power analysis attacks, but it has similar time and space complexities to them.

References

1. C. Clavier & M. Joye, *Universal Exponentiation Algorithm*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 300–308.
2. 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.
3. D. E. Knuth, *The Art of Computer Programming*, vol. **2**, "Seminumerical Algorithms", 2nd Edition, Addison-Wesley, 1981, 441–466.
4. 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.
5. 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.
6. D. May, H.L. Muller & N.P. Smart, *Random Register Renaming to Foil DPA*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 28–38.
7. T. S. Messerges, E. A. Dabbish & R. H. Sloan, *Power Analysis Attacks of Modular Exponentiation in Smartcards*, Cryptographic Hardware and Embedded Systems (Proc CHES 99), C. Paar & Ç. Koç (editors), Lecture Notes in Computer Science, **1717**, Springer-Verlag, 1999, 144–157.
8. 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.
9. C. D. Walter, *Exponentiation using Division Chains*, IEEE Transactions on Computers, **47**, No. 7, July 1998, 757–765.
10. C. D. Walter & S. Thompson, *Distinguishing Exponent Digits by Observing Modular Subtractions*, Topics in Cryptology – CT-RSA 2001, D. Naccache (editor), Lecture Notes in Computer Science, **2020**, Springer-Verlag, 2001, 192–207.
11. 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.
12. C. D. Walter, *Precise Bounds for Montgomery Modular Multiplication and Some Potentially Insecure RSA Moduli*, Topics in Cryptology – CT-RSA 2002, B. Preneel (editor), Lecture Notes in Computer Science, **2271**, Springer-Verlag, 2001, 30–39.
13. 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.